Detecting and mitigating the Distributed Denial of Service attacks in Software Defined Networks using Machine Learning approach – The Integrated Random Forest and K-Nearest Neighbours classifiers.

LEDWABA MATLALA

Student Number: [redacted]

A DISSERTATION SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF **MASTER OF SCIENCE IN COMPUTER SCIENCE**, DEPARTMENT OF **COMPUTER SCIENCE**, SCHOOL OF **MATHEMATICAL AND COMPUTER SCIENCES**, FACULTY OF **SCIENCE AND AGRICULTURE**, **UNIVERSITY OF LIMPOPO**, SOUTH AFRICA

SUPERVISOR: **Professor Mthulisi VELEMPINI**

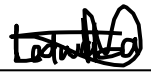
Sep 2023

DEDICATION

This dissertation is dedicated to my beloved people.
To my parents, Makwena Stephina Ledwaba and Matome Eric Ledwaba.
To my siblings Moloko, Tlou, Dineo, Mmaphuthi and Mmakgabo Ledwaba.
Lastly, to my late grandfather, Jacobus Lesiba Ledwaba.

DECLARATION

I, ***Matlala Ledwaba*** hereby declare that this dissertation entitled **Detecting and Mitigating the Distributed Denial of Service Attacks in Software Defined Networks using Machine Learning Approach – The Integrated Random Forest and K-Nearest Neighbours classifiers**, submitted to the University of Limpopo for a Master's degree, is my original work and has not been submitted previously to any university or institution of higher learning. I further declare that all sources cited are acknowledged and correctly referenced.

Signature:  Date: September 2023

ACKNOWLEDGEMENTS

I thank God for giving me the strength to complete this journey. He is indeed a keeper of his promises.

I appreciate the guidance I received from my supervisor, Professor Mthulisi Velempini. Thank you for your guidance, support, and encouragement throughout the process. Thank you for being patient and helping me stay focused and motivated. Getting feedback, comments, and advice from you has helped me grow tremendously. May God bless you.

My parents, Matome (my father) and Makwena (my mother) Ledwaba thank you for your support and love. I could not have done it without your encouragement. “Kodumela moepa thutse, ga go lehumo le le tswang kgauswi.”

To my siblings, Moloko, Tlou, Dineo, Mmaphuthi and Mmakgabo; kea leboga bana ba bomma. Thank you for telling me not to give up.

My best friend, Kholofelo Mabapa, you held my chin up when I needed you the most. Thank you.

Abstract

Distributed Denial of Service (DDoS) attacks present substantial risks to network availability and stability, especially within the realm of Software Defined Networks (SDNs). Inventive and efficient detection and mitigation methods become imperative to counter the continuously evolving nature of these attacks.

SDN is characterized by its dynamic and programmable nature and is susceptible to DDoS attacks that can disrupt network operations. Traditional methods for detecting and mitigating DDoS attacks in SDNs may not be sufficient due to the evolving nature of these attacks. The research aims to develop a more effective and adaptive solution by using the Random Forest (RF) and k-Nearest Neighbours (KNN) machine learning algorithms. This approach seeks to enhance the accuracy, speed, and resilience of DDoS detection and mitigation in SDN networks.

The research aims to address the pressing need for robust DDoS detection and mitigation mechanisms in SDNs by harnessing the power of machine learning, through the integration of RF and KNN and improving the KNN model. This approach is motivated by the evolving threat landscape, the unique challenges posed by SDN environments, and the potential for advanced machine learning techniques to enhance network security.

Furthermore, the research objective is to enhance the K-Nearest Neighbors (KNN) classification algorithm. By looking deep into KNN and addressing its limitations, this study seeks to refine and optimize the algorithm's performance for various real-world applications. Through a systematic exploration of parameter tuning, feature engineering, and innovative techniques, this research aims to provide a more accurate and efficient KNN classifier.

This study investigates the utilization of a machine learning approach, specifically Random Forest and K-Nearest Neighbours classifiers, to identify and counteract Distributed Denial of Service (DDoS) attacks in Software Defined Networks (SDNs). The research commences by exploring the fundamental concepts of SDNs and DDoS attacks, highlighting their interplay and the unique challenges they pose to network availability and stability.

The methodology for the study typically involves steps such as Data Collection (gathering network traffic data from SDN, including both normal and potentially malicious traffic.), Data Preprocessing (Clean and preprocess the collected data to remove noise, handle missing values, and normalize features), Feature Engineering: Identify relevant features or attributes in the network traffic data that can help distinguish between normal and DDoS attack traffic. By following the methodology presented in this study, we can systematically investigate the feasibility and efficacy of the proposed approach for detecting and mitigating DDoS attacks in SDN.

A comprehensive review of existing literature is conducted to understand the state-of-the-art techniques employed for DDoS detection and mitigation, with an emphasis on machine learning approaches. Expanding on the current understanding of mitigation against attacks, this thesis suggests employing Random Forest and K-Nearest Neighbours classifiers to improve the precision and effectiveness of DDoS detection in SDN environments. The proposed framework utilizes the ensemble learning abilities of Random Forest to address the challenges posed by the complex and diverse network traffic features, while the K-Nearest Neighbours algorithm offers the necessary flexibility and prompt decision-making for timely mitigation.

To evaluate the proposed model, extensive experiments are conducted using a realistic SDN simulator and diverse DDoS attack scenarios. Multiple performance metrics, including accuracy of detection, rate of false positives, and response time, are assessed and compared to alternative methods. The results demonstrate the superiority of the Random Forest and K-Nearest Neighbours classifiers in detecting and mitigating DDoS attacks effectively, efficiently, and with minimal impact on legitimate traffic.

In conclusion, this study shows that the improved KNN algorithm with a n_neighbours value of 2 has a higher accuracy rate compared to the Decision Tree classifier.

Furthermore, this research explores the challenges and limitations associated with the proposed model and provides insights for further improvements. This dissertation makes a valuable contribution to the domain of network security by introducing a novel methodology that employs machine learning techniques to identify and counteract DDoS attacks in SDNs. The model presented not only enhances the precision of attack detection but also diminishes response time, empowering network administrators to

safeguard their SDN infrastructure against intricate and evolving DDoS attacks effectively.

Keywords: Software-Defined Networks, Distributed Denial of Service Attacks, Machine Learning, Random Forest, k-Nearest Neighbours, Decision Tree, DDoS Detection, DDoS Mitigation, Network Security.

Table of Contents

List of Figures

DETECTING AND MITIGATING THE DISTRIBUTED DENIAL OF SERVICE ATTACKS IN SOFTWARE DEFINED NETWORKS USING MACHINE LEARNING APPROACH – THE INTEGRATED RANDOM FOREST AND K-NEAREST NEIGHBOURS CLASSIFIERS

CHAPTER 1: INTRODUCTION

Software-Defined Networks (SDNs) have revolutionized the networking landscape by enabling dynamic and programmable network control. However, this flexibility also introduces new security challenges, with Distributed Denial of Service (DDoS) attacks being one of the most prevalent and disruptive threats. DDoS attacks overwhelm network resources, rendering network services inaccessible to legitimate users. As traditional security mechanisms struggle to keep abreast with the sophistication and scale of these disruptive attacks, the integration of machine learning techniques has emerged as a promising approach to enhance DDoS detection and mitigation in SDNs. The enhancement of the K-NN classifier plays a huge role in DDoS detection and mitigation capabilities within SDNs.

Literature that deals with DDOS in SDNs:

The survey in [1] provides an overview of DDoS attack types and explores the potential of SDN in mitigating such attacks. It discusses various defence mechanisms and their effectiveness in SDN. The work in [2] evaluated DDoS detection and mitigation techniques in SDN. It compared traditional and SDN-based approaches, emphasizing the advantages and challenges of using SDN for DDoS protection.

The work in [3] focuses on various DDoS countermeasures specifically designed for SDN. It discusses proactive and reactive approaches and evaluates their effectiveness. These works offer valuable insights into the challenges and solutions related to DDoS attacks in SDN.

Proposed model:

This study focuses on the development and evaluation of an efficient machine learning-based model for detecting and mitigating DDoS attacks in SDNs. Specifically, we explore the effectiveness of two popular classifiers: Random Forest (RF) and k-Nearest Neighbours (k-NN). By leveraging the strengths of these classifiers, we aim to improve the accuracy, efficiency, and robustness of DDoS detection and mitigation in SDN environments.

The Random Forest classifier is an ensemble learning technique that builds numerous decision trees in parallel and combines their outputs to make predictions. It excels in handling high-dimensional datasets and provides robustness against noisy and irrelevant features. On the other hand, the k-Nearest Neighbours classifier is a non-parametric algorithm that classifies instances based on the similarity to their k-Nearest Neighbours. It is well-suited for detecting anomalies and can adapt to dynamic network conditions.

Why RF and K-NN

Using RF and k-NN classifiers offers the potential to leverage the complementary strengths of both algorithms, providing a more accurate and reliable detection system for DDoS attacks in SDNs. By training the classifiers on labelled network traffic data, the framework can learn patterns and behaviours associated with normal and malicious network traffic, enabling it to distinguish between legitimate and attacking flows.

RF and k-NN provide advantages for detecting and mitigating DDoS attacks in SDN due to their adaptability and ability to handle real-time data, low false positives, and scalability. When integrated with appropriate feature engineering and network monitoring techniques, these algorithms can significantly enhance the security of SDN networks against DDoS threats.

In this study, we present an in-depth development and evaluation of the proposed model using an SDN simulator and a comprehensive dataset of DDoS attacks. We assess the performance of the RF and k-NN classifiers in terms of detection accuracy, false positive rate, detection time, and resource utilization. We there enhance the performance of K-NN and thus improving K-NN's detection accuracy, false positive rate, detection time, and resource utilization. Additionally, the results of this study are compared to the Decision Trees algorithm results for the same dataset.

The findings of this research contribute to the development of effective DDoS detection and mitigation mechanisms in SDNs, providing network administrators and security professionals with enhanced capabilities to safeguard their infrastructures. These machine learning techniques hold great promise for addressing the evolving landscape of threats and ensure sufficient resilience of SDN environments against DDoS attacks.

It is important to note that the study primarily relied on an SDN dataset for its investigation. However, it is essential to clarify that the primary objective of the research was not the implementation of an SDN for the explicit purpose of detecting DDoS attacks. Instead, the study leveraged a machine learning approach, specifically the Integrated RF and k-NN algorithms, to enhance DDoS detection within an SDN context. The focus of this study was also on improving the K-NN machine learning classification so to improve the detection and mitigation of DDoS attacks. The SDN dataset served as a critical foundation for the research's data-driven analysis and development of machine learning models to address the DDoS threat in SDN environments.

By combining the RF ensemble learning technique for feature selection and robustness with the flexibility and adaptability of the k-NN algorithm for real-time anomaly detection, the integrated RF and K-NN model will significantly improve the accuracy and efficiency of DDoS attack detection and mitigation in SDN environments. Improving the K-NN classifier is essential because its refinement will significantly elevate the accuracy and responsiveness of DDoS attack detection and mitigation in Software-Defined Networks (SDNs).

This research has two hypotheses. Firstly, we hypothesize that the integration will achieve a lower false positive rate, higher detection accuracy, and faster response times compared to traditional DDoS mitigation techniques (Decision Tree). Additionally, the integration will perform better in adapting to evolving attack strategies and maintaining network performance, ultimately enhancing the security and resilience of SDN against DDoS threats.

Secondly, we hypothesize that enhancing the K-NN classifier's performance in detecting and mitigating Distributed Denial of Service (DDoS) attacks within Software-Defined Networks (SDNs) will result in improvement in attack detection accuracy and response efficiency, ultimately leading to a more robust and secure SDN infrastructure.

The dataset used for this study can be found in [4].

STATEMENT OF THE RESEARCH PROBLEM

The structure of SDN involves the segregation of the control plane and the data plane. By utilizing a centralized control plane (controller), network administrators can oversee

the entire network and exert control over its operations [5]. The separation of the data plane and control plane in SDN presents several benefits, but it also equally exposes SDN to security vulnerabilities, including the risk of Distributed Denial-of-Service (DDoS) attacks. DDoS floods the controller with traffic generated by a few compromised nodes. The controller becomes unavailable for useful services as a result of flooding which causes a single point of failure.

Although several DDoS countermeasures have been designed, there has been an increase in DDoS attacks in the SDN [6]. The promising DDoS countermeasure is conventional packet filtering (examining only packet headers), however, it cannot inspect the DDoS attacks hence there is a need for an accurate deep packet inspection (DPI) scheme designed to detect DDoS attacks.

RATIONALE

Software Defined Network (SDN) provides easy management, scalability and improved performance of cloud computing but if the SDN controller is not secured, then the SDNs are susceptible to DDoS attacks that result in the depletion of network bandwidth or the exhaustion of the victim's resources.

DDoS attacks are one of the commonly known network attacks in SDN [7]. As the number of network-connected devices increases, the impact of DDoS attacks becomes more pronounced [8]. There are several mechanisms which have been implemented to address the effects of DDoS. The schemes are designed to detect and reduce the effects of DDoS. Unfortunately, DDoS attacks persist in new technologies such as the SDN and CRN [9].

This study proposes a scheme which examines and mitigates DDoS attacks. The scheme implements a machine learning scheme equipped with the RF classifier and K-NN.

RELATED WORK

Random Forest is the algorithm used for predictive modelling. In predictive modelling, the variables are randomly selected to construct multiple decision trees [5]. In Random Forest, multiple decision trees are constructed from the given data set by repeatedly dividing the data set into subtrees (by changing the combination of variables) and the results are combined to make predictions [5]. The separation of the control plane and the data plane in SDN is designed to streamline the rapid deployment of new

resources with greater ease and efficiency [10]. Attackers can easily manipulate packet headers to camouflage malicious traffic as normal, evading detection systems. The attackers can simply modify the normal traffic since the anomalies are continuously evolving [10].

The proposal in [10] provides a detailed study of various approaches based on classic Machine Learning techniques that are used in detecting attacks in SDN. The study conducted a benchmarking experiment on the NSL-KDD dataset to evaluate the performance, highlighting the reasons why conventional machine learning methods struggle to achieve satisfactory results.

The study proposed in [10] uses the NSL-KDD dataset and provides more accurate results than using other datasets. Therefore, in our proposed study, the NSD-KDD dataset is used with the RF to get more accurate results on deep inspection of packets. Challenges that many machine learning algorithms face include the requirement of large amounts of data before they can begin to give useful results. The larger the architecture, the more data is needed to produce practicable results.

Today, Distributed Denial of Service (DDoS) attacks pose a significant and widespread threat to the Internet. DDoS attacks employ similar techniques to regular Denial of Service attacks, but on a much larger scale by utilising botnets. A botnet refers to a network of numerous compromised hosts (referred to as zombies, bots, or slave agents) under the control of one or more intruders, who orchestrate the attacks targeting specific victims [11].

There are many DDoS attacks and these attacks keep on increasing warranting a need for the detection schemes that should help to protect computers against DDoS attacks. In [12], the researcher discusses the current DDoS defence mechanisms and classifies them based on their primary functions, specifically detection, traceback, and mitigation. The researcher also discusses the strengths and weaknesses of the current DDoS defence mechanisms.

Scalability is a problem with most solutions, so defence mechanisms may not perform as expected in the real world. Networks are also burdened by significant extra computation and communication overhead even with the most current solutions. The inclusion of this additional overhead significantly affects network performance, leading

to substantial slowdowns in real-world scenarios where there is a high influx of attack traffic [12].

The study in [12] compared various mechanisms and found that not all solutions yield results for the necessary metrics used in comparison and therefore, conducting additional tests becomes necessary to evaluate their performance accurately using the chosen metrics.

Machine learning models are used to detect and mitigate DDoS attacks, but no detection model exhibits satisfactory detection accuracy due to the diversity of DDoS attack modes and the variable sizes of attacks [13]. Their study, [13] included feature extraction and model detection as components of a DDoS attack detection system based on machine learning. In the feature extraction stage, [13] illustrates that a significant portion of the traffic exhibited DDoS attack characteristics, which were determined by comparing and classifying the data packets using predefined rules. The extracted features were used for machine learning in the model detection stage, and the attack model was trained using a random forest algorithm. It was found by [13] that their suggested approach for detecting DDoS attacks, which utilizes machine learning, exhibits a robust detection rate.

The use of the Random Forest algorithm in [13] proved that even though most of the algorithms do not give satisfactory detection accuracy, there are still some algorithms (such as the random forest algorithm) that could be used to better detect the DDoS with a high level of accuracy.

The study conducted in [14] investigated feature selection and classification techniques for detecting Denial-of-Service (DoS) attacks, employing Random Forest (RF) for feature selection and k-Nearest Neighbours for classification. The findings confirmed that their proposed method achieved high accuracy in detecting both known and unknown attacks using the WEKA tool. Although the study in [14] demonstrated successful outcomes in DoS attack detection, further testing is necessary to assess its effectiveness against attacks like DDoS.

RESEARCH AIM

This study aims to develop an ensemble machine learning algorithm that integrates Random Forest and K-Nearest Neighbours to detect DDoS attacks in the SDN. This

study also aims to enhance the effectiveness of the K-Nearest Neighbours classifier as a DDoS attack detection and mitigation tool within Software-Defined Networks (SDNs), ultimately strengthening the security and resilience of SDN infrastructures against DDoS attacks.

RESEARCH OBJECTIVES

i. To investigate the effects of DDoS attacks in the SDN.
ii. To investigate the effectiveness of the Random Forest and K-NN classifiers.
iii. To optimize and improve the performance of the K-NN classifier.
iv. To design an integrated Random Forest and K-Nearest Neighbours classifiers security scheme.
v. To evaluate the effectiveness of the proposed integrated scheme in detecting and blocking the DDoS attack.

RESEARCH QUESTIONS

i. What are the effects of DDoS attacks on the performance of SDN?
ii. How can SDN be effectively protected from DDoS attacks?
iii. How can the performance of the K-NN classifier be effectively optimized and improved?
iv. How best can the Random Forest and KNN be used for the detection and blocking of DDoS attacks?
v. How effectively does the proposed scheme perform in the detection and mitigation of Distributed Denial of Service (DDoS) attacks?

RESEARCH HYPOTHESIS

The combination of the Random Forest and KNN classifiers can improve the detection and effectively block the DDOS from the SDN.

METHODOLOGY

In this research, POX is used because of the advantages stated below, instead of other different types of controllers such as Ryu, ONIX, Maestro, Beacon, etc.

POX is an OpenFlow/Software Defined Networking (SDN) Controller written in Python. It is a valuable tool for accelerating the development and prototyping of network applications. One of its advantages is that the POX controller is readily available as part of the pre-installed components in the Mininet virtual machine. By utilizing POX, it becomes feasible to transform standard OpenFlow devices into intelligent entities

like hubs, switches, load balancers, and firewalls. Moreover, the POX controller provides a user-friendly approach to conducting OpenFlow/SDN experiments, offering a convenient means to explore and evaluate various network scenarios [15].

RESEARCH SIMULATION

MININET

Mininet is widely recognised as a network emulator commonly utilised for SDN research purposes. It adopts a process-based virtualisation approach, enabling the execution of multiple hosts and switches within a single operating system kernel. By leveraging virtual hosts, switches, controllers, and links, Mininet offers the flexibility to create diverse network topologies. The hosts within Mininet operate on Linux network software, while the switches support OpenFlow, facilitating the development of OpenFlow-based applications within an SDN environment. Additionally, Mininet offers an extensible Python API that allows for the creation and customisation of networks.

TRAFFIC GENERATOR

The traffic generator Scapy is employed to generate UDP packets and manipulate the source IP address of the packets through spoofing.

PERFORMANCE METRICS OF MACHINE LEARNING APPROACH

The performance of our proposed detection system using the ML approach is evaluated using the parameters of accuracy, error, and precision. We use a confusion matrix to calculate these performance metrics.

PYTHON MACHINE LEARNING LIBRARY: SCIKIT-LEARN

Scikit-learn is capable of handling both supervised and unsupervised machine learning algorithms, offering a uniform, task-centric interface that facilitates a straightforward comparison of methods specific to a given application. In addition to traditional statistical data analysis, Scikit-learn can be easily integrated into applications outside of the scientific Python ecosystem [16].

DATA ANALYSIS AND LIBRARY: PANDAS

Pandas is a Python library that provides rich data structures and tools for working with structured data sets used in many different fields, including statistics, economics, and social sciences [17]. The library offers integrated procedures that are simple to use for common data manipulations and analysis on these kinds of data sets. This module is intended to serve as the foundation for Python's future statistical computing. Pandas implement and enhance the kinds of data manipulation features present in other

statistical programming languages like R, acting as a strong complement to the current scientific Python stack [17].

PLATFORM AND LANGUAGE

Linux

Python 3.8.0

DETECTING THE DDoS

In this study, for the detection of the DDoS, we use the Random Forest classifier and for blocking the DDoS, we use both the Random classifier and the KNN classifier.

BLOCKING

The following approach is used to block the DDoS after the detection of the DDoS is finished.

1. If the DDoS is detected, the algorithm identifies the DDoS features (using the RF) and blocks the DDoS packets.
2. If the DDoS is not detected, the algorithm goes back to the beginning of the process.

AVAILABILITY OF RESOURCES

Resources are available from open-access data and the University of Limpopo.

ETHICAL CONSIDERATION

The study does not require ethical clearance.

SCIENTIFIC CONTRIBUTION

Our study implements an Integrated Random Forest and K-Nearest Neighbours classifiers security scheme designed to mitigate the effects of DDoS in SDN. The scheme optimises the strengths of the Random Forest and the K-NN classifiers in an integrated environment. This study also enhances the performance of K-NN in detecting and mitigating DDoS attacks. This study also contributes to the body of knowledge in the field of security in SDN. This study makes a significant scientific contribution by advancing the state-of-the-art in machine learning and classification techniques, specifically in the context of optimizing and improving the performance of the K-NN classifier. The fine-tuning of various parameters, including the number of neighbours (k) and data preprocessing techniques, provide valuable insights into the factors that influence KNN's efficacy. This research equips both academia and

industry with powerful tools to improve the performance of K-NN across diverse domains, ultimately advancing the field of artificial intelligence and data science.

OVERVIEW OF THE STUDY

This dissertation is structured in five chapters. The initial chapter maps the research problem and provides sufficient context for the justification. Chapter two (2) is the literature review where we examine the problems of DDoS in the SDN and how machine learning techniques differ. Chapter three (3) provides details on the research methodology, and we assess the simulation environment, scenarios considered, and model training and testing. Chapter four (4) presents the results and discussion where we look at the comparison of results between the proposed scheme and other (1) existing algorithms. Chapter five (5) concludes the study and in it, we make a final remark based on our findings and proffer recommendations for future research.

CHAPTER 2: LITERATURE REVIEW

INTRODUCTION

The objective of this chapter is to present an extensive review encompassing the current research, theories, methodologies, and findings about the detection and mitigation of DDoS attacks in Software Defined Networks (SDNs) utilising machine learning approaches.

This chapter establishes the solid groundwork for the research study by providing a thorough synthesis of pertinent and recent literature. Through a detailed examination of prior studies, methods, and results, the chapter identifies existing gaps, challenges, and potential avenues for further exploration in the field of mitigating DDoS attacks in SDNs. Moreover, it lays the foundation for the theoretical framework and conceptual understanding that guides the subsequent chapters and the development of the integrated model proposed and developed in this study.

Furthermore, this chapter conducts an extensive and detailed examination of the current research regarding machine learning approaches employed for the detection and mitigation of DDoS attacks. It explores a range of algorithms, techniques, and methods utilised in previous studies, offering insights into their respective strengths, effectiveness and identifying their limitations. The review encompasses both supervised and unsupervised learning algorithms, emphasising their practical applicability and performance in addressing the unique challenges presented by DDoS attacks within SDN environments.

Through this comprehensive literature review, the chapter establishes the research context, identifies gaps in existing knowledge, and lays the foundation for the proposed approach that leverages the Random Forest and K-Nearest Neighbours classifiers. By building upon the established body of knowledge, this study contributes to the advancement of DDoS detection and mitigation strategies within SDN environments.

LITERATURE REVIEW

Software Defined Network

SDN is an emerging type of network that significantly simplifies network management tasks. SDN has a programmable flexible interface that controls the behaviour of the entire network [18]. Instead of deploying a distributed control architecture, SDN consolidates all control functions into a centralised entity known as the 'Network

Controller.' The Network Controller is software that runs on a commercial server platform.

In the SDN, the control and data planes are separated. This separation helps in having the control plane negotiation differentiated from the node that handles the end user traffic. This aspect simplifies the network manageability and programmability [18] [19].

In their study, [20] overviewed the SDN architecture, that is, its current and future applications; [20] presented a study of networks looking at the motives and challenges of SDN. The history of programmable networks, from early ideas until recent developments, is discussed. Their work includes describing the SDN in detail as well as the OpenFlow standard. The current SDN implementations were presented, platforms were tested, and they also examined the network services and applications that have been developed based on the SDN paradigm. [20] concluded by examining future directions facilitated by SDN, which encompass various aspects such as the provision of heterogeneous network support and the adoption of Information-Centric Networking (ICN).

Although SDN the control plane enables enhanced control over network entities, SDN becomes a burden on the administrator because the same administrator must manually ensure security and correct the functioning of the whole network [21]. In their study, [21] listed several attacks on SDN controllers that disregard the network topology and data plane forwarding and can be from compromised network entities.
[21] proposed SPHINX framework to detect both familiar and unfamiliar attacks in the context of SDN. SPHINX is designed to adaptively learn and identify new network behaviours, triggering alerts whenever it detects suspicious changes in the network [21]. SPHINX can detect attacks in SDNs in real-time with low-performance overheads and it requires no changes to the controller for deployment [21]. Existing controllers are vulnerable to known and/or unknown attacks and SPHINX can effectively detect them in real-time [21]. SPHINX comes with minimal overheads [21].

Numerous studies have been conducted to evaluate and compare the efficiency, features, and architecture of different SDN controllers [22]. A library bundle called Libfluid offers the fundamental capabilities needed to construct an OpenFlow controller [22]. In Open Network Operating System (ONOS), bundles are written in Java, and they are loaded into the Karaf OSGi container [22]. The Linux Foundation is the home

of the OpenDaylight (ODl) initiative, a Java-based collaborative open-source initiative. Different non-OpenFlow southbound protocols are supported by the OpenDaylight (ODl), which also allows bidirectional REST and OSGi framework programming Java [22]. A networking software platform called POX (Pythonic Network Operating System) was created using the Python programming language and can be useful when building networking software [22]. Open SDN controller Ryu Controller was created to improve network agility.

The Ryu framework is a Python component-based software defined networking framework [22]. The controllers can give a throughput and delay response while increasing the load on the linear topology [22]. In the test, a linear topology was built in the Mininet emulator with a different number of switches. The results show that, among those five controllers, the POX controller gives the best delay performance and the Libfluid controller gives the best throughput performance. Although POX and Libfluid gave better results, it is important to choose the best-performing controller based on several criteria, according to the requirements of the user.

The authors in [23] evaluated the scalability of the Floodlight Controller using Mininet, Floodlight Controller and iPerf. In a study by [23], the performance evaluation of the Floodlight Controller was conducted within a simulation environment. The study focused on monitoring the throughput and latency parameters of the controller. To simulate dynamic networking conditions, the performance of the controller was assessed using a Mesh topology, with the number of nodes exponentially increasing. Besides providing simulation experimental test bed support, Floodlight controllers also provide statistical analysis after simulation experiments have been conducted [23]. They did not compare the controllers of the SDN and thus recommend it as future research work.

A custom component for the POX controller platform that helped in overcoming the infinite loop problem using Python was created by authors in [24]. Their proposed study can be applied to other available OpenFlow controllers with different APIs.

Distributed Denial of Service attacks

DDoS is a network threat that exhausts network resources to make them unavailable to legitimate users [25]. The DDoS violates the “availability” component of cyber security. The attacker launches the attack using multiple computers and subsequently

compromises the different systems using mechanisms such as Trojans, worms, etc. The compromised systems are called Zombies and the controller system is called a master [25]. These Zombies can be situated across the globe and thus one may not differentiate them from the legitimate traffic [25].

Previous Work done

The effectiveness of common systems like IPS and IDS in detecting and preventing DDoS attacks is often limited when it comes to new attack signatures or previously unseen attack techniques [26]. Utilising machine learning and pattern recognition, novel forms of DDoS attacks can be analysed and mitigated seamlessly, ensuring uninterrupted protection [26]. In their study, [26] a comprehensive survey was conducted on DDoS attacks, focusing on the utilisation of data mining techniques to identify DDoS attack patterns and analyse these patterns using machine learning algorithms. In their study, [26] also highlighted open issues, research challenges and possible solutions in machine learning.

Below is a table of the findings from the study conducted by [26]

Functions / Research	Detect DDoS Attack Patterns	Analyze Patterns By Machine Learning	KNN	SVM	Random Forest	Naïve Base	Block MAC Address of Victims
Detecting Distributed Denial of Service (DDoS) Attacks through Inductive Learning	✓	✓	x	x	x	x	x
The Design of the Network Service Access Control System through Address Control in IPv6 Environments	x	x	x	x	x	x	✓
Proactively Detecting Distributed Denial of Service Attacks Using Source IP Address Monitoring	✓	x	x	x	x	x	x
A Survey of Distance and Similarity Measures Used Within Network Intrusion Anomaly Detection	x	✓	✓	✓	x	x	x
Network Intrusion detection using Naïve Bayes	x	x	x	x	x	✓	x
Evaluating Machine learning Algorithms for detecting Network Intrusion	x	✓	x	x	✓	✓	x
An Ann Approach for network Intrusion Detection using Entropy Based Feature Selection	x	x	x	✓	x	x	x
Automatic Analysis of Malware Behavior using machine learning	x	✓	x	✓	x	x	x
Detecting And Blocking unauthorized Access in Wi-Fi networks	x	x	x	x	x	x	✓
Detecting distributed Denial of service attacks	✓	x	x	✓	x	x	x
Detecting DDOS Attacks Against webserver via Lightweight TCM-KNN Algorithm	x	x	✓	x	x	x	x
DOS attack detection Based on naïve Bayes classifier	x	x	x	x	x	✓	x
Detection model for denial of service attack using Random Forest and K-Nearest Neighbors	x	x	✓	✓	✓	x	x
Detection of low and High rate DDOS attack using matrix with SVM in fireCol Distributed Network	x	x	x	✓	x	x	x
Machine learning Techniques used in detection of DOS attacks	x	✓	x	✓	x	✓	x
Investigation of DHCP packets using Wireshark	x	x	x	x	x	x	✓
A covariance Analysis model for DDOS attack detection	✓	x	x	x	x	x	x
Detecting Denial of service attacks with incomplete audit data	x	x	✓	x	x	x	x
Detecting denial of service attacks with Bayesian classifiers and the random Neural networks	x	x	x	x	x	✓	x
DDOS attacks detection and prevention using ensemble classifier(Random forest)	✓	x	✓	x	✓	x	x
Improved Support Vector Machine for cyber-attack detection	x	x	x	✓	x	x	x
Preventing DDoS attack using Data mining	✓	✓	✓	x	x	✓	✓

Figure 1 Summary of findings from a study conducted by [21]

The aforementioned table displays data mining algorithms' highest accuracy rate for averting DDoS assaults. [21] investigated leading machine learning algorithms such as KNN, Support Vector Machine (SVM), Random Forest as well as Naïve Base for detection of DDoS. The study conducted by [26] focused on identifying DDoS attacks based on UDP flooding.

As technology advances, the SDN becomes more vulnerable to DDoS attacks and there is a need to protect the SDN from such attacks. This can be done using the machine learning algorithms such as the Random Forest classifier and the KNN classifier. The Random Forest classifier randomly selects features, or combinations of features, at each node during its decision-making process [27].

The research conducted by [27] had the objective of assessing and comparing the performance of the Random Forest classifier and Support Vector Machines (SVMs) in terms of classification accuracy, training time, and the number of user-defined parameters. The results obtained from their investigation established similar performance between the Random Forest classifier and SVMs in terms of classification accuracy and training time. However, it was observed that the Random Forest classifier required fewer user-defined parameters compared to SVMs, making it easier to define [27]. These findings suggest that the Random Forest classifier holds potential as a competitive alternative to SVMs, offering comparable performance with simpler parameter configuration.

Only two parameters need to be set for the Random Forest classifier, compared to several user-defined parameters for SVMs [27]. In contrast to SVMs, the Random Forest classifier is capable of effectively handling categorical data, imbalanced data, and missing values in the dataset [27]. The relevance of various features throughout the classification process is also provided by the Random Forest classifier, which is helpful in feature selection [22]. The Random Forest classifier can be used for unsupervised learning and offers a way to identify outliers [27].

Random Forests are versatile and can be applied to handle both categorical responses for classification tasks and continuous responses for regression tasks [28].

Random Forests have the following advantages [28]:

- Random Forest classifiers can naturally handle both regression and classification
- Training and prediction are relatively fast
- There are only a few tuning parameters to consider
- Provide a built-in estimate of generalization error
- High-dimensional problems can be solved directly through this method
- Parallel implementation is possible
- Variable importance measurements
- A differential weighting system for classes
- Missing value imputation
- Visualisation
- Outlier detection
- Unsupervised learning

An in-depth analysis of Random Forests was provided in [29], where a comprehensive analysis of each component of the algorithm is performed to gain fresh insights into the learning capabilities, internal mechanisms, and interpretability of Random Forests. Authors in [29] divided their work into three parts.

In the initial phase, a study by [29] examined the methodology of decision trees and Random Forests concerning classification and regression tasks. Their research aimed to develop a unified and adaptable framework by considering the induction of individual decision trees. [29] thoroughly examined assignment rules, stopping criteria, and splitting rules, offering theoretical justification for the design and purpose of decision trees whenever possible. Additionally, they established a correlation between variance and individual tree predictions, emphasising how randomisation reduces generalisation error. The study also presented and discussed the Random Forest algorithm and its variations within the established framework, highlighting their properties and features. Furthermore, [29] provided an original analysis of the computational complexity of Random Forests, demonstrating their notable performance and scalability for larger problems that were observed, which led to a thorough discussion of implementation details in the initial part of their research, emphasising critical considerations for ensuring an optimal computational performance that is often overlooked.

In part two of their work, [29] analysed and discussed the interpretability of Random Forests by looking at variable importance measures. The analysis established that variable importance offers a thorough assessment of the three-tier breakdown of information conveyed by the input variables regarding the output, encompassing all potential interaction terms exhaustively. In addition [29] demonstrated that variable importance is solely influenced by relevant variables, with the importance of irrelevant variables consistently measured as zero. This consolidates the validity and appropriateness of using "importance" as a criterion for assessing variable utility. Furthermore, the second part of the study confirmed that variable importance is susceptible to certain limitations stemming from masking effects, wrong estimation of node impurity, and the binary structure inherent to decision trees.

The final third section of the study [29] discussed the limitations of Random Forests when applied to large datasets. It was found that subsampling, either in terms of samples, features, or both concurrently, allows for consistent performance while reducing memory demands.

We should consider machine learning as a method that emphasizes logical thinking and customization to address specific problems, rather than treating it as a black-box tool [29].

In the previous studies about how to identify and address the DDoS from the SDN, no study was done using the integration of both the Random Forest and the KNN classifiers. Also, no study was done on improving K-NN for the detection and mitigation of DDoS attacks in SDNs. This study integrates the Random Forest and the KNN classifier to detect and mitigate the DDoS from the SDN. This study improves K-NN for the detection and mitigation of DDoS attacks in SDNs.

CONCLUSION

In this literature review, we explored the topic of detecting and mitigating Distributed Denial of Service (DDoS) attacks in Software-Defined Networks (SDNs) using a machine learning approach, specifically focusing on the using Random Forest (RF) and k-Nearest Neighbours (k-NN) classifiers. Through an extensive analysis of relevant studies and research articles, several key findings and insights emerged.

Firstly, the prevalence and severity of DDoS attacks in SDNs have been well-established. The flexible and adaptable characteristics of SDNs introduce new security

challenges, requiring innovative approaches to effectively detect and mitigate these attacks. Traditional security mechanisms often fall short of keeping pace with the rapidly evolving attack techniques and large-scale attack volumes. This necessitates the exploration of advanced solutions, such as machine learning, to enhance DDoS defence in SDNs.

The review of existing literature emphasised how machine learning techniques have proven effective in enhancing the detection and mitigation of DDoS attacks within SDN environments. Various machine learning algorithms have been investigated, each with their strengths and limitations. Among them, Random Forest and k-Nearest Neighbours classifiers have emerged as preferred choices due to their complementary capabilities and applicability to SDN security.

Random Forest classifiers are demonstrably robust in handling high-dimensional data, making them suitable for DDoS detection in SDNs, where the feature space can be complex and dynamic. Additionally, the ensemble nature of Random Forest classifiers allows for improved accuracy and generalisation, enabling effective identification of anomalous network traffic patterns associated with DDoS attacks.

On the other hand, k-Nearest Neighbours classifiers are capable of identifying similarities between instances, making them adept at detecting deviations from normal network behaviour. Their non-parametric nature and adaptability to changing network conditions make them valuable tools for anomaly detection in SDN environments.

By using Random Forest and k-Nearest Neighbours classifiers, we leverage their respective strengths and address the limitations of individual algorithms. This approach holds promise for enhancing the accuracy, efficiency, and robustness of DDoS detection and mitigation in SDNs.

Based on the insights gained from the literature review, our research endeavours to add to the current understanding and body of knowledge in this field by conducting an in-depth evaluation of the Random Forest and k-Nearest Neighbours classifiers. Through extensive experimentation using an SDN simulator and comprehensive datasets of DDoS attacks, our objective is to evaluate the success of the recommended approach by analysing its detection accuracy, false positive rate, detection time, and resource utilisation. Furthermore, comparative analyses with other

commonly used machine learning algorithms (Decision Trees) for DDoS detection in SDNs are conducted to provide a comprehensive evaluation.

In conclusion, the literature review shed light on the significance of detecting and mitigating DDoS attacks in SDNs and the potential of machine learning techniques, particularly the Random Forest and k-Nearest Neighbours classifiers, to enhance SDN security. The subsequent chapters of this study build upon these findings, providing empirical evidence and insights that contribute to the development of effective DDoS defence mechanisms in SDNs.

CHAPTER 3: METHODOLOGY

INTRODUCTION

In Software-Defined Networking (SDN), the identification and prevention of Distributed Denial of Service (DDoS) attacks are critical practices as these malicious activities have the potential to disrupt the network's operation. This chapter aims to present the methodology that was adopted in the research conducted to identify and counteract DDoS attacks within SDN. This section presents and justifies the chosen research design, methods employed for data collection, techniques utilised for data analysis, as well as the tools implemented in this study.

The study focuses on detecting and mitigating Distributed Denial of Service (DDoS) attacks in Software Defined Networks (SDNs) using a Machine Learning approach, specifically employing the Integrated Random Forest and K-Nearest Neighbours classifiers. The goal of the study is to improve K-NN and therefore enhancing the security of SDNs by developing a system that accurately identifies and mitigates DDoS attacks in real time.

We describe the methodology used to train and evaluate the classifiers. We outline the process of collecting and pre-processing network traffic data, including feature extraction to represent the network traffic patterns effectively. The dataset used encompasses both normal and attack traffic, allowing the classifiers to learn the distinguishing characteristics of DDoS attacks.

Training the classifiers involves splitting the dataset into training and testing sets. The study outlines the training procedure for both classifiers, including parameter selection and optimisation techniques. It discusses the evaluation metrics used to assess the classifiers' performance, such as accuracy, True-positives, False-positives and the Receiver Operating Characteristic (ROC) curve analysis.

Both the training and detection phases are implemented within the SDN controller. In data collection, network traffic data, including flow information, packet headers, and network statistics, is gathered from SDN switches and forwarded to the centralized SDN controller. While in data pre-processing, the collected data is pre-processed to clean and transform it into a suitable format for machine learning. This step involves handling missing data, normalizing features, and converting categorical variables into numerical ones.

In Feature Extraction and Selection, relevant features are extracted from the pre-processed data to represent network traffic behaviour effectively while in Machine Learning Model Training, the integrated RF and K-NN machine learning models are trained on historical network data within the SDN controller. During training, the models learn to distinguish between normal network behaviour and DDoS attack patterns.

On the other hand, model integration**,** once trained, the machine learning models are integrated into the SDN controller's logic. This integration allows the controller to make real-time decisions based on the predictions made by the models while with detection, as network traffic flows through the SDN switches, the SDN controller continuously monitors the traffic patterns. It uses the integrated machine learning models to detect any anomalies or patterns consistent with DDoS attacks.

Lastly, mitigation is when a potential DDoS attack is detected, the SDN controller can take various mitigation actions. These actions might include isolating the affected traffic, rerouting it through specific paths, or implementing rate limiting to mitigate the attack's impact.

The methodology employed to optimize and enhance the performance of the K-NN classifier involves a systematic and iterative approach. Initially, a comprehensive review of the existing literature is conducted to identify best practices, challenges, and recent advancements in KNN optimization. Data preprocessing is a fundamental step, where feature selection and extraction techniques are applied to enhance the quality and relevance of input data. Subsequently, an exhaustive hyperparameter tuning process is undertaken, leveraging techniques like fine-tune KNN's parameters, such as the number of neighbors (k).

Throughout the experimentation process, performance metrics such as accuracy, False positives, False negatives, True positives, True negatives and receiver operating characteristic (ROC) curves are monitored and analysed.

In this study, we then compare the performance of the proposed approach to existing methods and showcase the superiority of the Machine Learning approach in terms of accuracy, speed, and robustness.

The chapter concludes with a discussion of the results, highlighting the successful detection and mitigation capabilities of the proposed approach. It emphasises the potential of Machine Learning algorithms in enhancing the security of SDNs against DDoS attacks. This study also discusses the limitations and potential future directions for improving the proposed approach.

SIMULATION ENVIRONMENT

This study used the comparison and benchmarking methodology to compare the performance of our proposed model with existing DDoS detection and mitigation methods or benchmarks.

In this study, a simulation environment is essential to assess and measure the effectiveness of the suggested approach. Based on this simulation environment, testing and evaluation are carried out.

The simulation environment is designed to mimic a real Software Defined Network (SDN) environment and generate network traffic that simulates normal and attack traffic. The environment can generate various forms of DDoS attacks, including TCP, UDP, and ICMP and test the effectiveness of the proposed approach in detecting and mitigating these attacks.

The dataset used in this study is collected from Kaggle [4] where they used a network emulator called Mininet. These emulators provide a virtual network environment where one can configure network topologies, network traffic, and devices. The simulation environment can create an SDN topology with OpenFlow switches, OpenFlow controllers, and SDN-enabled devices such as SDN firewalls.

To assess the effectiveness of the proposed approach, performance metrics such as accuracy, false-positive rate, true-positive rate, Receiver Operating Characteristic (ROC) curve, and Area Under the Curve (AUC) are employed. These metrics help evaluate the ability of the proposed approach to detect and mitigate DDoS attacks.

SCENARIOS CONSIDERED

In this study, we take into account several scenarios to comprehensively evaluate the proposed solution. These scenarios are:

1. Normal Network Traffic: Simulate and analyse the performance of the proposed approach under normal network conditions. This scenario helps establish a

baseline for comparison and ensures that our approach does not generate false positives or misclassify legitimate network traffic.

2. Various Types of DDoS Attacks: Test the solution's ability to detect and mitigate different types of DDoS attacks, such as TCP, UDP and ICMP. Each attack type may have different traffic patterns and characteristics, and the approach should be robust enough to identify and respond to them effectively.
3. Real-time Detection and Mitigation: Test the solution's ability to detect and respond to DDoS attacks in real time. Assess the impact on network performance and user experience.
4. Performance Comparison: Compare the performance of the proposed approach with the performance of the Decision Tree classifier to evaluate their effectiveness in detecting and mitigating DDoS attacks. Assess metrics such as accuracy, true-positive rate and false-positive rate to determine which approach yields robust results.

TRAINING

In this study, training involves several steps such as:

1. Data Collection: Gather a labelled dataset that includes network traffic data, both normal and attack traffic. This dataset should represent various types of DDoS attacks and normal network behaviour. The data collection is performed by using publicly available datasets from Kaggle.
2. Data Pre-processing: Pre-process the collected data to prepare it for training the classifiers. This involves removing irrelevant features, handling missing values, normalising the data, and balancing the dataset if there is a class imbalance issue. Pre-processing ensures that the data is in a format suitable for training the classifiers. Data Pre-processing is done in a tool called Jupyter Notebook.
3. Feature Extraction: Extract relevant features from the pre-processed data that capture the characteristics of network traffic. These features include packet header information, flow statistics, traffic patterns, and protocol-specific attributes. Careful selection and engineering of features can significantly impact the classifiers' performance.
4. Dataset Split: Divide the pre-processed data into separate training and testing sets. The training set is utilised to train the classifiers, whereas the testing set

is used to evaluate their performance. The data is divided into two parts (25% test dataset and 75% training dataset). Figure 2 shows part of the dataset.

```
dataset = dataset.iloc[:70000,:]
dataset
```

	dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	...	pktrate	Pairflow	Protocol	port_no	tx_bytes	r
0	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451	0	UDP	3	143928631	
1	11605	1	10.0.0.1	10.0.0.8	126395	134737070	280	734000000	2.810000e+11	2	...	451	0	UDP	4	3842	
2	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	1	3795	
3	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	2	3688	
4	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	3	3413	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
69995	7128	2	10.0.0.1	10.0.0.7	112958	127971940	264	337000000	2.640000e+11	4	...	449	1	TCP	3	5139477	12
69996	7128	2	10.0.0.1	10.0.0.7	112958	127971940	264	337000000	2.640000e+11	4	...	449	1	TCP	2	5287	
69997	7128	2	10.0.0.7	10.0.0.1	77614	5123220	264	302000000	2.640000e+11	4	...	335	1	TCP	1	5139	
69998	7128	2	10.0.0.7	10.0.0.1	77614	5123220	264	302000000	2.640000e+11	4	...	335	1	TCP	4	130058363	
69999	7128	2	10.0.0.7	10.0.0.1	77614	5123220	264	302000000	2.640000e+11	4	...	335	1	TCP	3	5139477	12

70000 rows × 23 columns

Figure 2 Dataset used for this work

5. Training the Random Forest Classifier: Train the Random Forest classifier using the training data. A Random Forest method combines multiple decision trees into one ensemble learning method. A different subset of data and features is used to train each tree. The ultimate determination is reached by combining the classifications from all the trees. During training, the classifier learns to identify patterns and distinguish between normal and attack traffic.
6. Training the K-Nearest Neighbours Classifier: Utilise the training data to train the K-Nearest Neighbours (KNN) classifier. KNN is a classification algorithm that operates on the principle of proximity, assigning class labels to samples based on their proximity to training instances. It determines the class of a test sample by considering the majority class among its ***k*** nearest neighbours. During training, the KNN classifier learns the distances and relationships between different instances in the feature space.
7. Model Evaluation: Evaluate the performance of the trained classifiers using the testing set. Calculate evaluation metrics such as accuracy, True-positives, False-positives and ROC-AUC to assess their effectiveness in detecting and mitigating DDoS attacks.

Mininet is a platform that enables the simulation of extensive network prototypes on a single computer [30]. It utilises virtualisation techniques, including processes and network namespaces, to facilitate the creation of scalable Software-Defined Networks (SDNs). Mininet offers the ability to rapidly create, interact with, customize, and share prototypes [30].

Mininet can create SDN elements such as hosts, switches, controllers and links. Those elements can be customised, shared with other networks and perform interactions [30]. This study uses the Mininet to create a scheme that effectively detects and mitigates the DDoS attack (UDP) in the SDN. The SDN has virtual switches, virtual hosts, virtual links and virtual controllers.

The traffic is generated so that the SDN can be tested to establish if it can differentiate the right packets from the DDoS packets. To generate the traffic, we utilise the Scapy traffic generator to create UDP packets and manipulate the source IP address of each packet for spoofing purposes. Python programming languages are used in this study to build the Random Forest and the KKN classifier for detection and mitigation of the DDoS attack.

The Pandas library is a Python archive that offers efficient and user-friendly data structures as well as data analysis tools for Python programs [31]. Pandas are used in this study for data analysis.

To evaluate the performance of our proposed system using the ML approach, the following parameters will be taken into consideration:

i. Accuracy: the degree to which the results of the proposed system conform to the expectations of how a good system for detection and mitigation of the DDoS should perform.
ii. Error: the error rate of the proposed system compared to the systems that are not using the combination of the Random Forest classifier and KNN. The time that our proposed scheme takes to detect and mitigate the DDoS is compared to the time that the other schemes take to detect and mitigate the DDoS.

Integration Challenges faced when integrating RF and K-NN:

1. RF is an ensemble learning method that operates based on decision trees, while K-NN is an instance-based learning algorithm. These algorithms have distinct characteristics and decision-making processes. Integrating them requires finding a way to coordinate their outputs and decision strategies. This need more time and research on how to do it.
2. Finding tools that can handle diverse data sources, formats, and quality while aligning with both Random Forest and KNN requirements is challenging.

Overcoming these challenges is time-consuming, as it may involve custom development, adaptation of existing tools, and extensive testing. This delays the implementation of the DDoS detection and mitigation system.

The tools used in this study are all open-source tools and are available for use in this study. The Python programming language is used to write the code for the proposed scheme in the Mininet.

CONCLUSION

In conclusion, the methodology adopted for the detection and mitigation of DDoS attacks in SDN is crucial for ensuring the protection of the network. The use of a systematic research design, reliable data collection methods and effective data analysis techniques greatly enhanced the accuracy and efficacy of the research. The adaptation of appropriate tools and techniques for carrying out the research is vital in identifying potential DDoS attacks and ensuring the mitigation of these attacks. The methodology informed the research process and ultimately helped to achieve the desired results in detecting and mitigating DDoS attacks in SDN.

CHAPTER 4: RESULTS AND DISCUSSION

INTRODUCTION

This chapter analyses and interprets the outcomes of the implemented methodology. It further explores their implications in the context of combating DDoS attacks in Software Defined Networks (SDNs).

The chapter provides an in-depth examination of the results generated, evaluating the effectiveness and performance of the proposed model. Furthermore, it discusses the significance and implications of the findings in addressing the ever-growing challenge of DDoS attacks.

The chapter is structured as follows: first, the presentation of the experimental setup and dataset used for evaluation; second, the analysis of the detection performance of the classifiers; and finally, a comprehensive discussion of the observed results and their implications.

The discussion and interpretation of these findings contribute to the existing body of knowledge in the field of network security. These findings also serve as a basis for further advancements in the detection and mitigation of evolving DDoS threats.

In this section, we present the results obtained from evaluating the two proposed Machine Learning models (Random Forest and K-Nearest Neighbouring algorithm) on the dataset. The results are compared to those obtained when evaluating the Decision Tree algorithm.

The basic steps of the two Machine Learning algorithms can be summarised as follows:

1. Collect raw data (From Kaggle. DDoS SDN dataset)
2. Process the data
3. Feature selection
4. Create sub-dataset
5. Train K-Nearest Neighbouring (KNN) algorithm
6. Calculate the accuracy of (KNN)
7. Train Random Forest (RF) algorithm
8. Calculate the accuracy of RF.

The dataset (DDoS SDN dataset) for classification is downloaded from Kaggle [4].

We cleaned and analysed the dataset to train the two Machine Learning models used in this research.

The following Machine Learning models are trained in this study:

1. Random Forest (RF)
2. K-Nearest Neighbouring algorithm (K-NN).

After training the above two models, we set to improve the KNN model.

Python Libraries import, dataset evaluation and cleaning

We used Jupiter Notebook for all the work done in this study. Pandas, Numpy and other Python libraries are imported and used. We cleaned and evaluated the dataset so that it would be ready for use for the two models developed in this study.

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import seaborn as sns

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import preprocessing

from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn import metrics
```

Figure 3 Importing Python Libraries

The dataset had 104 345 rows and 23 columns. The amount of the dataset was too much for the models to train and we ran into memory errors and therefore reduced the dataset to 70 000 as shown below.

```
dataset = pd.read_csv('dataset_sdn.csv')
dataset
```

	dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	...	pktrate	Pairflow	Protocol	port_no	tx_bytes	rx_by
0	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451	0	UDP	3	143928631	3
1	11605	1	10.0.0.1	10.0.0.8	126395	134737070	280	734000000	2.810000e+11	2	...	451	0	UDP	4	3842	3
2	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	1	3795	1
3	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	2	3688	1
4	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	3	3413	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
104340	5262	3	10.0.0.5	10.0.0.7	79	7742	81	842000000	8.184200e+10	5	...	0	0	ICMP	1	15209	12
104341	5262	3	10.0.0.5	10.0.0.7	79	7742	81	842000000	8.184200e+10	5	...	0	0	ICMP	3	15099	14
104342	5262	3	10.0.0.11	10.0.0.5	31	3038	31	805000000	3.180500e+10	5	...	1	0	ICMP	2	3409	3
104343	5262	3	10.0.0.11	10.0.0.5	31	3038	31	805000000	3.180500e+10	5	...	1	0	ICMP	1	15209	12
104344	5262	3	10.0.0.11	10.0.0.5	31	3038	31	805000000	3.180500e+10	5	...	1	0	ICMP	3	15099	14

104345 rows × 23 columns

Figure 4 Full Dataset Rows and Columns

```
dataset = dataset.iloc[:70000,:]
dataset
```

	dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	...	pktrate	Pairflow	Protocol	port_no	tx_bytes	r
0	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451	0	UDP	3	143928631	
1	11605	1	10.0.0.1	10.0.0.8	126395	134737070	280	734000000	2.810000e+11	2	...	451	0	UDP	4	3842	
2	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	1	3795	
3	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	2	3688	
4	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	3	3413	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
69995	7128	2	10.0.0.1	10.0.0.7	112958	127971940	264	337000000	2.640000e+11	4	...	449	1	TCP	3	5139477	12
69996	7128	2	10.0.0.1	10.0.0.7	112958	127971940	264	337000000	2.640000e+11	4	...	449	1	TCP	2	5287	
69997	7128	2	10.0.0.7	10.0.0.1	77614	5123220	264	302000000	2.640000e+11	4	...	335	1	TCP	1	5139	
69998	7128	2	10.0.0.7	10.0.0.1	77614	5123220	264	302000000	2.640000e+11	4	...	335	1	TCP	4	130058363	
69999	7128	2	10.0.0.7	10.0.0.1	77614	5123220	264	302000000	2.640000e+11	4	...	335	1	TCP	3	5139477	12

70000 rows × 23 columns

Figure 5 Reduced Dataset Rows and Columns

From the reduced dataset, we have 43.3% of packets which are normal and 56.7% of malicious packets as shown below.

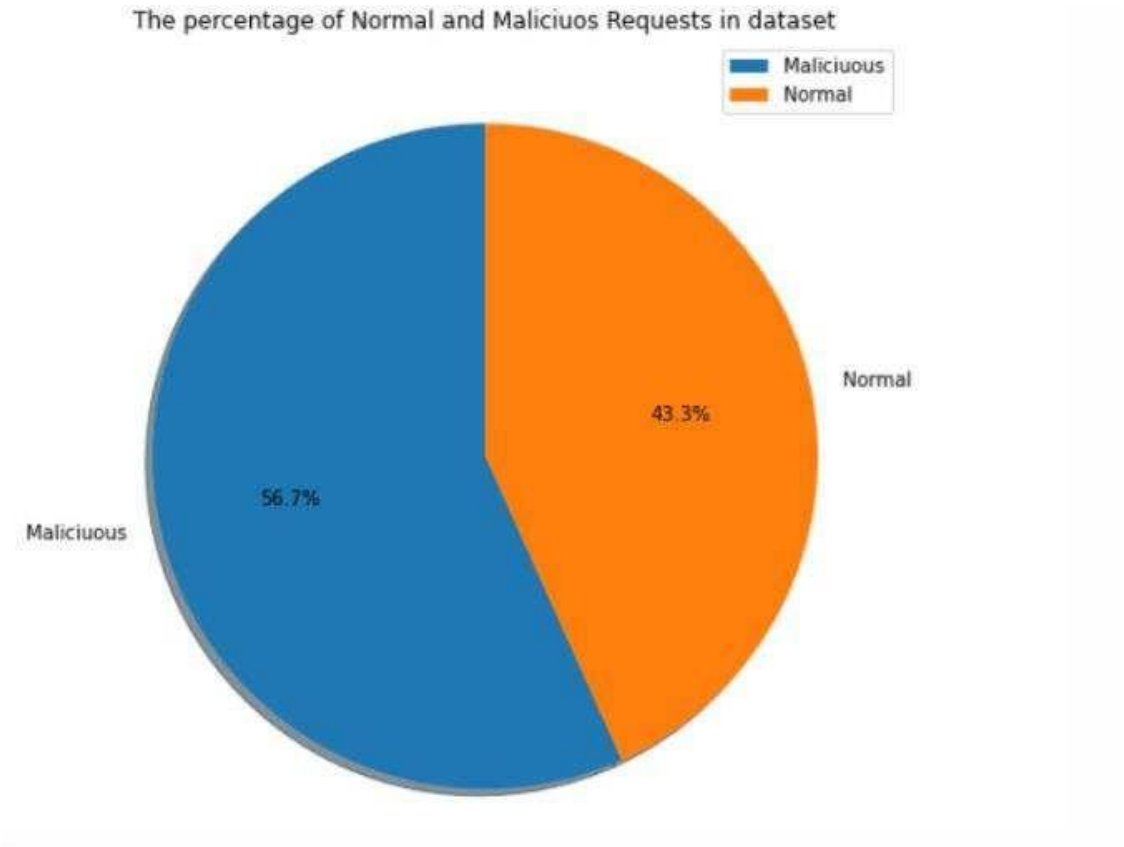


Figure 6 Normal and Malicious Packets

The packets were sent from 19 IP addresses with each sender having several requests.

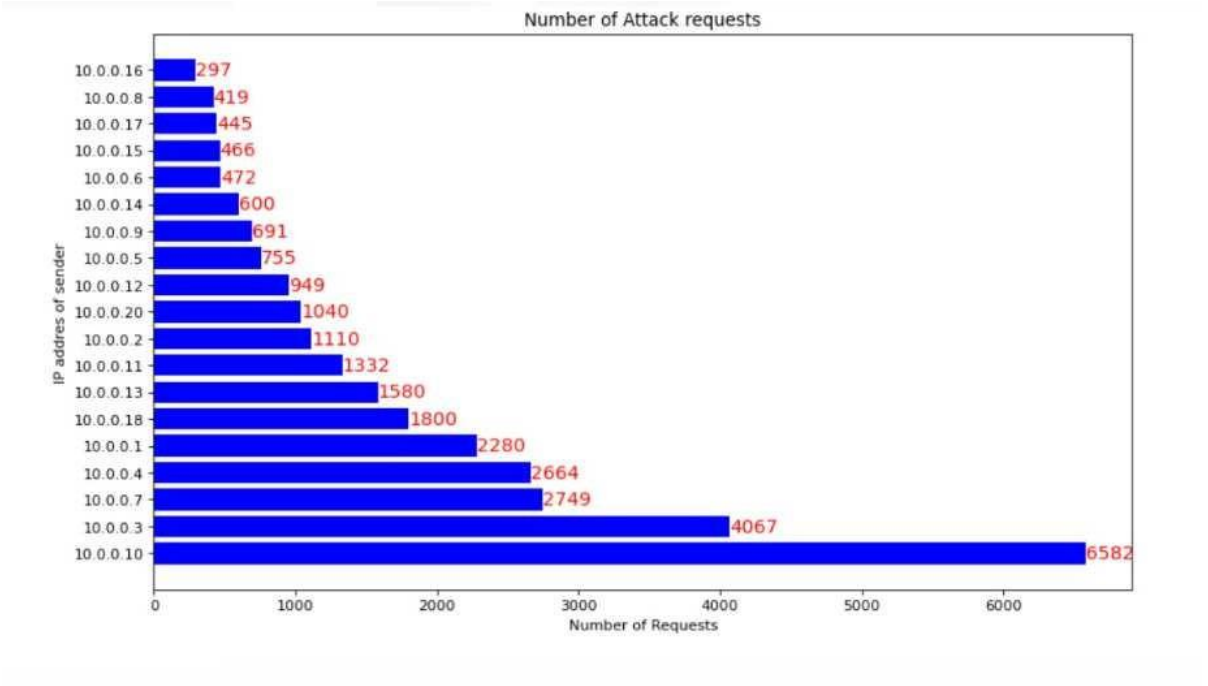


Figure 7 Number of Packets per IP address of the sender

For our dataset to be correctly trained, all entries had to be integers. We transformed the protocol column to numeric values as shown in Figure 7:

```
Y = dataset.iloc[:,11].values
Y
```

```
array(['UDP', 'UDP', 'UDP', ..., 'TCP', 'TCP', 'TCP'], dtype=object)
```

```
from sklearn.preprocessing import LabelEncoder
labelencoder_y = LabelEncoder()
Y = labelencoder_y.fit_transform(Y)
Y
```

```
array([2, 2, 2, ..., 1, 1, 1])
```

Figure 8 Changing protocols names to numeric values

Model (KNN) training

Our dataset is divided into two distinct parts (25% test dataset and 75% training dataset). As KNN depends on n-neighbours for training, we trained our model using 5 n-neighbours.

MODEL (KNN) TRAINING

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25, random_state = 5)
```

```
X_train.shape
```

```
(52500, 6)
```

```
X_train
```

```
array([[    31543,         7,     80096,   5286636,       260, 420000000],
       [    11575,         3,    112761, 120203226,       250, 305000000],
       [     9876,         2,     19494,  20780604,        42, 894000000],
       ...,
       [     2740,         1,      2671,   2847286,         5, 651000000],
       [     9786,         6,       972,   1012824,         3, 452000000],
       [    11349,         2,     38892,   2567028,       124, 349000000]],
      dtype=int64)
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, Y_train)
```

```
KNeighborsClassifier()
```

Figure 9 Importing KNeighboursClassifier

Sensitivity, specificity, accuracy, Cohen's kappa and confusion matrix were used for our model evaluation.

A confusion matrix is a valuable method utilised to provide an overview of a classification model's performance. It presents the count of correctly classified and misclassified instances. Table 1 shows the confusion matrix table.

	Actual: Yes	Actual: No
Predicted: Yes	True Positives(TP)	False Positives(FP)
Predicted: No	False Negatives(FN)	True Negatives(TN)

Table 1 Confusion matrix

In this study, we consider two classes: 'attack' and 'normal.' Within the confusion matrix, the columns represent the actual classes, while the rows represent the predicted classes. By analysing the confusion matrix, we can determine the number of correctly and incorrectly predicted results by the model.

Accuracy is a statistical metric employed to assess the proportion of accurate predictions, encompassing both true positives and true negatives. [32]. The formula for determining accuracy is as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP, TN, FP, and FN mean the true positives, true negatives, false positives, and false negatives, respectively. In training KNN, we strive to achieve an accuracy rate of at least 99%. Our model results in 95% accuracy which shall then be improved later in this chapter.

	precision	recall	f1-score	support
0	0.94	0.93	0.93	5807
1	0.94	0.84	0.89	3353
2	0.95	1.00	0.97	8340
accuracy			0.95	17500
macro avg	0.94	0.92	0.93	17500
weighted avg	0.94	0.95	0.94	17500

Figure 10 KNN Results when n_neighbors = 5

Random Forest Training

Random Forest is trained using 50 n-estimators and entropy as a criterion. In training the Random Forest classifier, we want to achieve an accuracy of at least 99%. Our Random Forest (which is trained as shown on figure 10) model resulted in 100% accuracy as shown on figure 11:

RANDOM FOREST TRAINING

```
Y_pred = classifier.predict(X_test)
Y_pred
len(Y_train)
```

```
52500
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
RFclassifier = RandomForestClassifier(n_estimators = 50, criterion = 'entropy')
RFclassifier.fit(X_train, Y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=50)
```

Figure 11 Importing Random Forest Classifier

	precision	recall	f1-score	support
0	1.00	1.00	1.00	17424
1	1.00	1.00	1.00	9828
2	1.00	1.00	1.00	25248
accuracy			1.00	52500
macro avg	1.00	1.00	1.00	52500
weighted avg	1.00	1.00	1.00	52500

Figure 12 Random Forest results

Improving KNN

From the results in figure 9, we used 5 n-neighbours and achieved 95% accuracy. To improve the Model such that it achieves at least 99% accuracy, we reduce n-neighbours. As we already trained the model using 5 n-neighbours, to improve the model, we start from 4-neighbours going down until we get 99% accuracy.

n_neighbors = 4

In [33]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
classifier = KNeighborsClassifier(n_neighbors = 4, metric = 'minkowski', p = 2)
classifier.fit(X_train, Y_train)
```

Out[33]:
```
KNeighborsClassifier(n_neighbors=4)
```

Figure 13 Training KNN with n_neighbours = 4

Sensitivity and specificity are both essential metrics for evaluating the performance of a model. Sensitivity measures the ability of a test to accurately identify true positives [33]. The formula for determining sensitivity is as follows:

$$Sensitivity = \frac{number\ of\ true\ positives}{number\ of\ true\ positives\ +\ number\ of\ false\ negatives}$$

where the count of true positives signifies accurate classification of positive classes, while the count of false negatives denotes misclassification of negative classes, i.e., their real label is a positive class, but they are classed as a negative class.

Specificity is a measure of how successfully a test identifies true negatives [33]. The formula for determining specificity is as follows:

$$Specificity = \frac{number\ of\ true\ negatives}{number\ of\ true\ negatives\ +\ number\ of\ false\ positives}$$

The count of true negatives represents the accurate classification of negative classes, while the count of false positives indicates the misclassification of positive classes, meaning that their true label is negative, but they are mistakenly classified as positive.

As shown in Figure 12, when n_neighbours=4, we achieve 99% sensitivity and 87% specificity. That is, KNN correctly predicted 99% of true positives and 87% of true negatives. KNN managed to correctly predict 32.18% of the UDP protocol,16.64% of the ICMP protocol and 47.64% of the TCP protocol. All these predictions resulted in 96% accuracy.

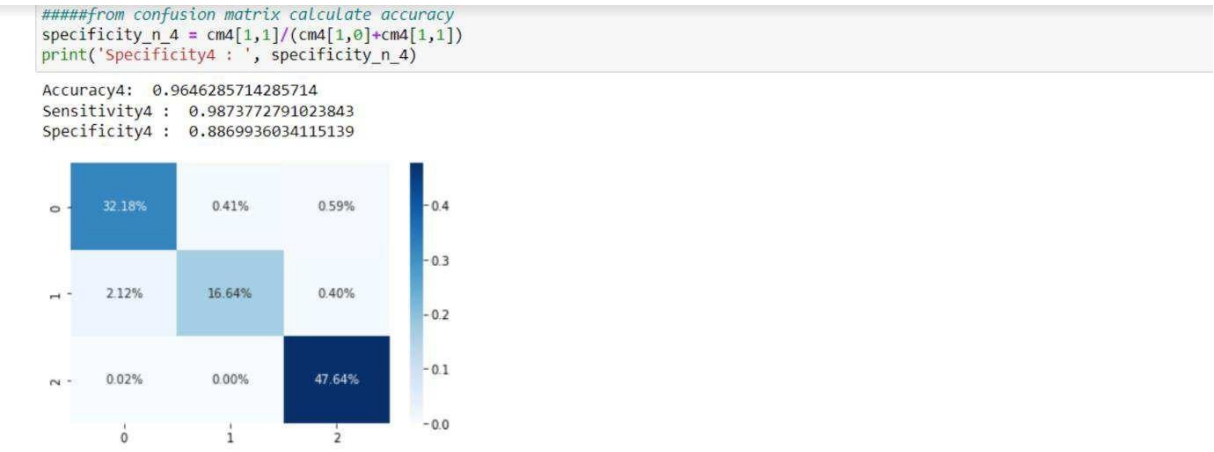


Figure 14 KNN Accuracy, Sensitivity and Specificity when n_neighbors = 4

To see how the model performed, we calculated the true positives and true negatives of the model for each n-neighbours. We calculated these values for each protocol. As shown in Figure 15 below, we have 3 true negative rates; for UDP, ICMP and TCP respectively. The same applies to positive predicted values, false positive values and false negative values.

```
print('TPR: ', TPR)
print('Specificity or true negative rate: ', TNR)
print('Precision or positive predictive value: ', PPV)
print('Negative predictive value: ', NPV)
print('Fall out or false positive rate: ', FPR)
print('False negative rate: ', FNR)
print('Overall accuracy for each class: ', ACC)
```

```
TPR:  [0.96986396 0.86847599 0.99964029]
Specificity or true negative rate:  [0.96801505 0.99491058 0.98111354]
Precision or positive predictive value:  [0.93772894 0.97587131 0.97967098]
Negative predictive value:  [0.98477467 0.96961973 0.9996663 ]
Fall out or false positive rate:  [0.03198495 0.00508942 0.01888646]
False negative rate:  [0.03013604 0.13152401 0.00035971]
Overall accuracy for each class:  [0.96862857 0.97068571 0.98994286]
```

Figure 15 True/False positives and Negatives when n_neighbors = 4

We trained the model with n_neighbours=3 also.

As shown in Figure 15, when n_neighbours=3, we achieve 98.65% sensitivity and 95.46% specificity. That is, KNN correctly predicted 98.65% of true positives and 95.46% of true negatives. KNN managed to correctly predict 32.18% of the UDP protocol,17.91% of the ICMP protocol and 47.64% of the TCP protocol. All these predictions resulted in 97.72% accuracy.

n_neighbors = 3

```
In [41]: from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import classification_report, confusion_matrix
         classifier = KNeighborsClassifier(n_neighbors = 3, metric = 'minkowski', p = 2)
         classifier.fit(X_train, Y_train)
Out[41]: KNeighborsClassifier(n_neighbors=3)
```

Figure 16 Training KNN with n_neighbours = 3

```
#####from confusion matrix calculate accuracy
specificity_n_3 = cm3[1,1]/(cm3[1,0]+cm3[1,1])
print('Specificity : ', specificity_n_3)
```

```
Accuracy :  0.9772571428571428
Sensitivity :  0.9865101611772951
Specificity :  0.954614681693573
```

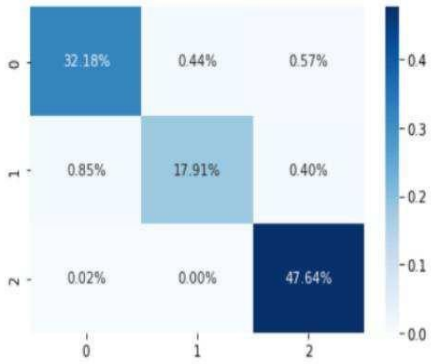


Figure 17 KNN accuracy, Sensitivity and Specificity when n_neighbors = 3

To see how the model performed when n_neighbours=3, we calculated the true positives and true negatives of the model. We calculated these values for each protocol. As shown in Figure 16 below, we have 3 true negative rates; for UDP, ICMP and TCP respectively. The same applies to positive predicted values, false positive values and false negative values.

```
print('TPR: ', TPR)
print('Specificity or true negative rate: ', TNR)
print('Precision or positive predictive value: ', PPV)
print('Negative predictive value: ', NPV)
print('Fall out or false positive rate: ', FPR)
print('False negative rate: ', FNR)
print('Overall accuracy for each class: ', ACC)
```

```
TPR:  [0.96969175 0.93468536 0.99964029]
Specificity or true negative rate:  [0.98700077 0.99455715 0.98155022]
Precision or positive predictive value:  [0.97371606 0.97601993 0.98013167]
Negative predictive value:  [0.98497909 0.98467353 0.99966644]
Fall out or false positive rate:  [0.01299923 0.00544285 0.01844978]
False negative rate:  [0.03030825 0.06531464 0.00035971]
Overall accuracy for each class:  [0.98125714 0.98308571 0.99017143]
```

Figure 18 True/False positives and Negatives when n_neighbours = 3

We also trained the model with n_neighbours=2. As shown in Figure 15, when n_neighbours=2, we achieve 99.89% sensitivity and 95.15% specificity. That is, KNN correctly predicted 99.89% of true positives and 95.15% of true negatives. KNN managed to correctly predict 32.18% of the UDP protocol,17.91% of the ICMP protocol and 47.64% of the TCP protocol.

When n-neighbours is 2, the model achieved 99% accuracy as shown in Figure 18 below. Therefore, we achieved what we set out to do for this study.

n_neighbors = 2

```
In [49]: from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import classification_report, confusion_matrix
         classifier = KNeighborsClassifier(n_neighbors = 2, metric = 'minkowski', p = 2)
         classifier.fit(X_train, Y_train)

Out[49]: KNeighborsClassifier(n_neighbors=2)
```

Figure 19 Training KNN with n_neighbours = 2

```
#####from confusion matrix calculate accuracy
specificity_n_2 = cm2[1,1]/(cm2[1,0]+cm2[1,1])
print('Specificity : ', specificity_n_2)
```

```
Accuracy :  0.9888
Sensitivity :  0.9989646246764452
Specificity :  0.951497005988024
```

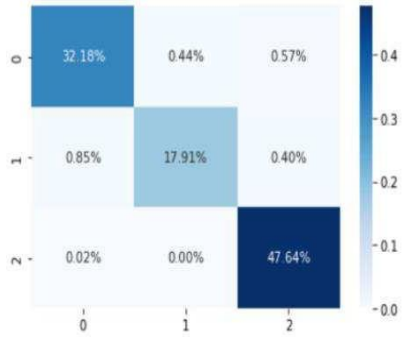


Figure 20 KNN Accuracy, Sensitivity and Specificity when n_neighbours = 2

In machine learning, the performance evaluation of a model ensures that it accurately predicts outcomes on new data. An extensively utilised approach to evaluate how well a classification model performs involves determining the counts of true positives, true negatives, false positives, and false negatives for every class or label existing in the dataset. In our research, we examined the performance of a model with a n_neighbours value of 2 utilising this technique.

To calculate the true positives and true negatives, we used a dataset that contained different protocols and their corresponding labels. We subsequently computed the count of accurately classified instances, considering both true positives and true

negatives, for each protocol. Figure 19 below shows the true negative rates for UDP, ICMP and TCP, respectively.

The true negative rate is the proportion of negative instances (instances where the predicted label was negative and the true label was also negative) that were correctly classified by the model. Similarly, we computed the true positive rate, which represents the percentage of positive cases (where the predicted label was positive and the actual label was also positive) that the model accurately classified.

Furthermore, we calculated the false positive and false negative rates for each protocol. The false positive rate represents the percentage of negative cases that were mistakenly classified as positive by the model. Conversely, the false negative rate signifies the proportion of positive instances that were inaccurately labelled as negative by the model.

By calculating these performance metrics, we were able to assess the effectiveness of the model with n_neighbours=2 in accurately classifying instances for each protocol.

```
print('Precision or positive predictive value: ', PPV)
print('Negative predictive value: ', NPV)
print('Fall out or false positive rate: ', FPR)
print('False negative rate: ', FNR)
print('Overall accuracy for each class: ', ACC)
```

```
TPR:  [0.99690029 0.94780793 0.99964029]
Specificity or true negative rate:  [0.98588899 0.99957588 0.99727074]
Precision or positive predictive value:  [0.97228754 0.99811558 0.99701028]
Negative predictive value:  [0.99844102 0.98777592 0.9996717 ]
Fall out or false positive rate:  [0.01411101 0.00042412 0.00272926]
False negative rate:  [0.00309971 0.05219207 0.00035971]
Overall accuracy for each class:  [0.98954286 0.98965714 0.9984    ]
```

Figure 21 True/False positives and Negatives when n_neighbours = 2

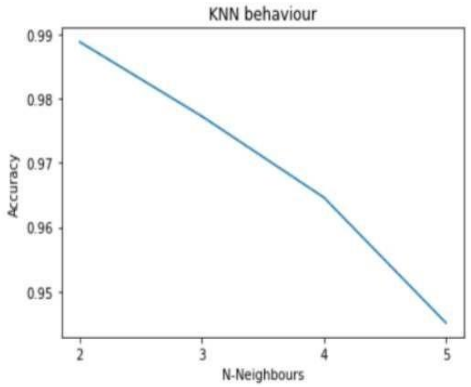
In the field of machine learning, the selection of hyperparameters plays a crucial role in determining the accuracy of a model. For example, when employing the k-nearest neighbour (KNN) algorithm, the decision regarding the number of neighbours to consider has a substantial influence on the overall performance of the model. To

identify the optimal number of neighbours, it is standard practice to assess the model's accuracy across various values of this hyperparameter.

This research investigated how the accuracy of the KNN algorithm is affected by different numbers of neighbours. We tested the algorithm with different values of n-neighbours and calculated the corresponding accuracy rates. Our evaluation results showed that as the number of neighbours increased, the accuracy of the model decreased.

This trend can be attributed to the observation that as the quantity of neighbours increases, the algorithm considers more data points when making predictions. However, including more neighbours also increases the likelihood of noise or outliers affecting the precision of the forecasts. Furthermore, it is important to recognise that increasing the number of neighbours can lead to overfitting, a situation where the model becomes excessively tailored to the training data and struggles to generalise well to new, unfamiliar data. Figure 23 shows that the more we increase n-neighbours, the more the accuracy decreases.

```
# function to show the plot
plt.show()
```




```
[61]: y_accuracy
[61]: [0.9888, 0.9772571428571428, 0.9646285714285714, 0.9451428571428572]
```

Figure 22 Improved KNN accuracy for when n-neighbours = 2,3,4 and 5

The Decision Tree classifier is a popular machine learning algorithm employed for classification purposes. In this research, we utilised the Decision Tree model to analyse a dataset and assess its accuracy. Our evaluation results (in Figure 24) show that the Decision Tree classifier achieved an accuracy rate of 95%. However, this

accuracy rate is less desirable compared to the 99% accuracy achieved when using the K-Nearest Neighbour (KNN) algorithm on the same dataset.

While the Decision Tree model is often preferred for its interpretability and ease of use, it may not always be the optimal choice for complex datasets or problems with a large number of features.

```
[28]: from sklearn.tree import DecisionTreeClassifier
      from sklearn import metrics

[29]: classifier = DecisionTreeClassifier(criterion="entropy", max_depth=3)

      classifier = classifier.fit(X_train,Y_train)

      Y_pred = classifier.predict(X_test)
      Decision_Tree_accuracy = metrics.accuracy_score(Y_test, Y_pred)

      print("Accuracy:",Decision_Tree_accuracy)

      Accuracy: 0.9532571428571428
```

Figure 23 Decision Tree Results

PDF Curve

Probability Density Functions (PDFs) describe the likelihood that a continuous random variable will take on different values. It provides a consistent way to represent the probability distribution of a random variable.

The PDF is defined such that the integral (area under the curve) of the function over a given range corresponds to the probability of the random variable falling within that range. In other words, the PDF generates information about the likelihood of the random variable taking on specific values or falling within certain intervals.

Figure 24 below shows the probability distribution of our proposed scheme and the comparative scheme (decision tree).

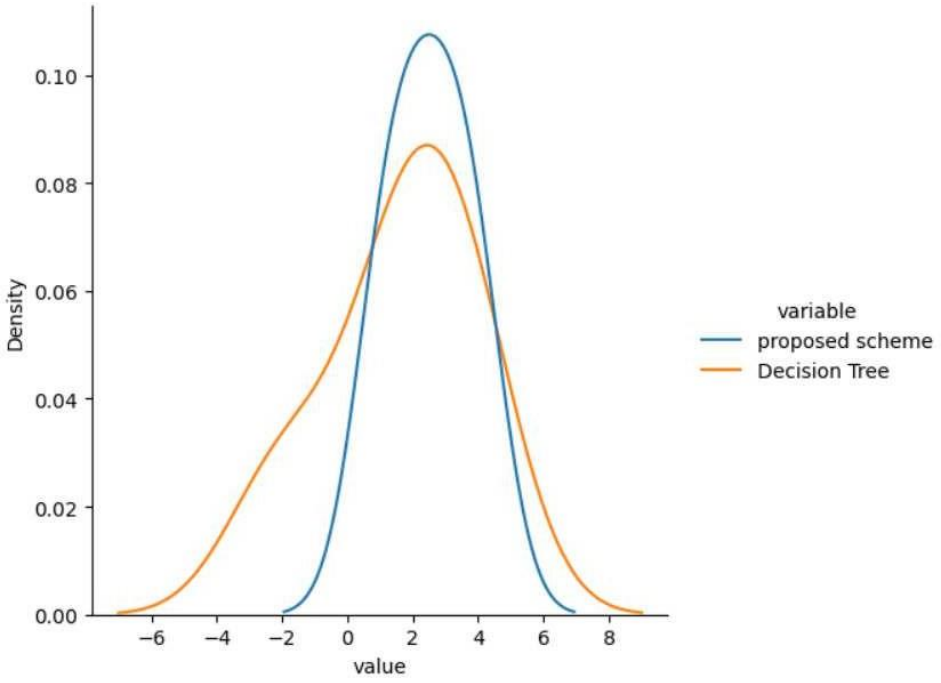


Figure 24 PDF Curves

For our proposed scheme, we conducted calculations to determine the probability of the range P (1 < X < 3) based on a normal distribution with a mean of 0 and a standard deviation of 3. The result of our analysis indicates that the cumulative probability of this range is 99%.

This finding implies that if we were to randomly select values from the aforementioned normal distribution, there is a strong likelihood of 99% that the selected values would fall within the range of 1 to 3. In other words, the probability of observing a value within this specific range is highly significant, reinforcing the reliability and predictability of the proposed scheme. Such a high probability allows us to have confidence in the consistency and stability of the data within this range, providing valuable insights for decision-making and analysis within our scheme.

The PDF value obtained is the density of the distribution within the range 1 < x < 3 which corresponds to the cumulative probability of 99%. The results obtained indicate that the model demonstrates a high level of accuracy in correctly identifying positive instances (true positives) within the specified range (1 < x < 3). Additionally, the model effectively minimizes the occurrence of false positives, ensuring reliable classification outcomes.

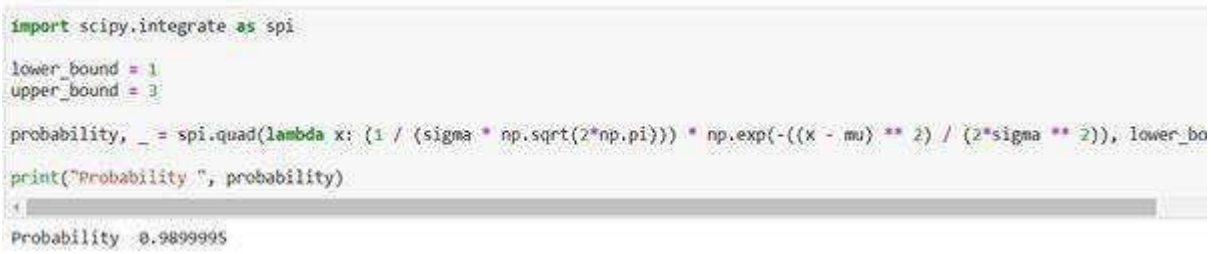


Figure 25 PDF results for our proposed model

This indicates how densely the distribution is populated within this range. However, it should be noted that the PDF does not represent the actual probability of specific values occurring, but rather the relative likelihood.

In practical terms, this result suggests that if one were to randomly select values from a Gaussian distribution with an average of 0 and a standard deviation of 3, there is a high probability (99%) that the selected values would fall within the range of 1 to 3.

For the comparative scheme, we conducted the same calculations to determine the probability of the range P (1 < X < 3) based on a normal distribution with a mean of 0 and a standard deviation of 3. The result of our analysis indicates that the cumulative probability of this range is 85%.

Figure 26 PDF results for the comparative scheme

Based on the results obtained from our proposed scheme and the comparative scheme, we can conclude that our proposed scheme yielded a PDF P (1 < x < 3) with a cumulative probability of 99% when considering a normal distribution with a mean (μ) of 0 and a standard deviation (σ) of 3. This indicates a very high likelihood that a randomly selected value from this distribution falls within the range of 1 to 3. The density of data points within this range is significant, and we can have a high level of confidence in the consistency and stability of the data within this interval. The

proposed scheme demonstrates strong predictability and reliability in terms of the data distribution within the specified range.

In comparison, the comparative scheme resulted in a PDF P ($1 < x < 3$) with a cumulative probability of 85% under the same conditions of a mean of 0 and a standard deviation of 3.

Therefore, it is advisable to use our proposed scheme which yielded a desirable likelihood.

ROC Curve

A ROC (Receiver Operating Characteristic) curve serves as a visual representation of a binary classification model's performance. It illustrates the relationship between the true positive rate (TPR) and the false positive rate (FPR) at various threshold values.

The TPR corresponds to the ratio of correctly identified positive instances by the model, while the FPR represents the ratio of negative instances mistakenly classified as positive. The threshold value acts as a boundary to differentiate or classify the positive and negative classes.

We can calculate different TPR and FPR values, and plot them on the ROC curve. The AUC (Area Under the ROC Curve) serves as a widely used metric to assess the performance of a classification model. A value of 1 signifies a flawless classifier, whereas an AUC of 0.5 represents a classifier that performs randomly. A higher AUC indicates superior classification performance.

The ROC curve is valuable for comparing the effectiveness of various classifiers or different parameter configurations within the same classifier. It visually illustrates the model's capacity to differentiate between positive and negative classes, aiding in the selection of an optimal threshold value for a specific classification task.

Figure 25 and Figure 26 below show the ROC curve of our model. In Figure 26, we see how our model performed because of the red dotted line below the graph. As the ROC curve approaches the top left corner of the plot, our model's performance in classifying the data improves.

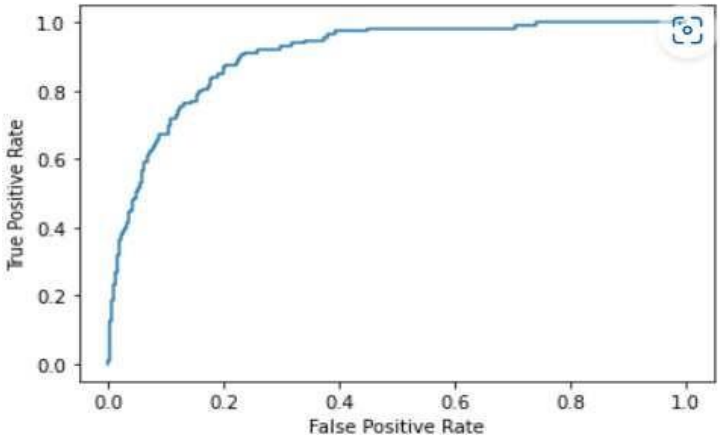


Figure 27 ROC Curve

The effectiveness of a classification model is commonly evaluated using the metric area under the Receiver Operating Characteristic (ROC) curve. The ROC curve is a graphical representation of the model's performance, with the area under the curve (AUC) quantifying the overall performance. AUC values range from 0 to 1, where higher values indicate better performance. An AUC of 0.5 represents an average classifier, while an AUC of 1 represents a perfect classifier. An AUC of 0 suggests that the classifier is always incorrect, while an AUC of 0.5 suggests that the classifier performs no better than random chance. By calculating the AUC, we determined the proportion of the curve that lies under the plot, providing a measure of the classifier's performance.

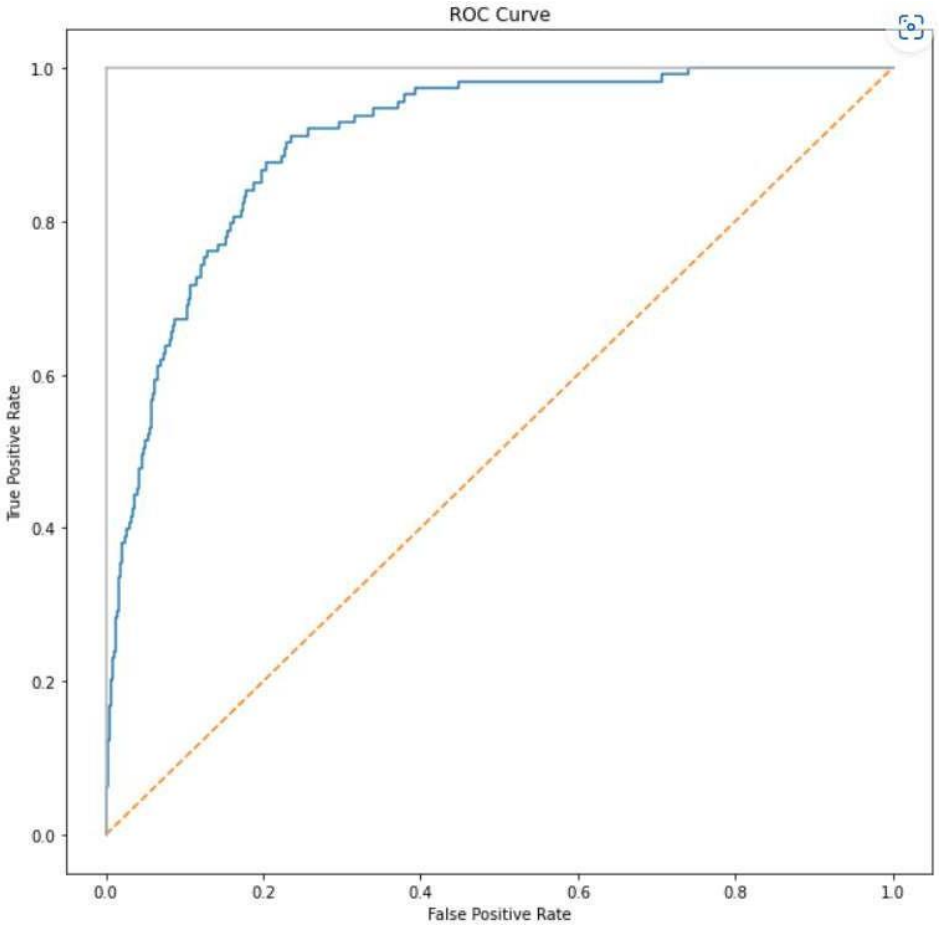


Figure 28 Clear ROC Curve for our proposed scheme

```
print('roc_auc_score: ', roc_auc_score(y_test, y_pred_proba))
```

```
roc_auc_score:  0.9050028967204219
```

Figure 29 AUC Score for our proposed scheme

As shown in Figure 28 and Figure 29, our model has the highest AUC (0.905), which indicates that it correctly classifies observations and has the highest area under the curve.

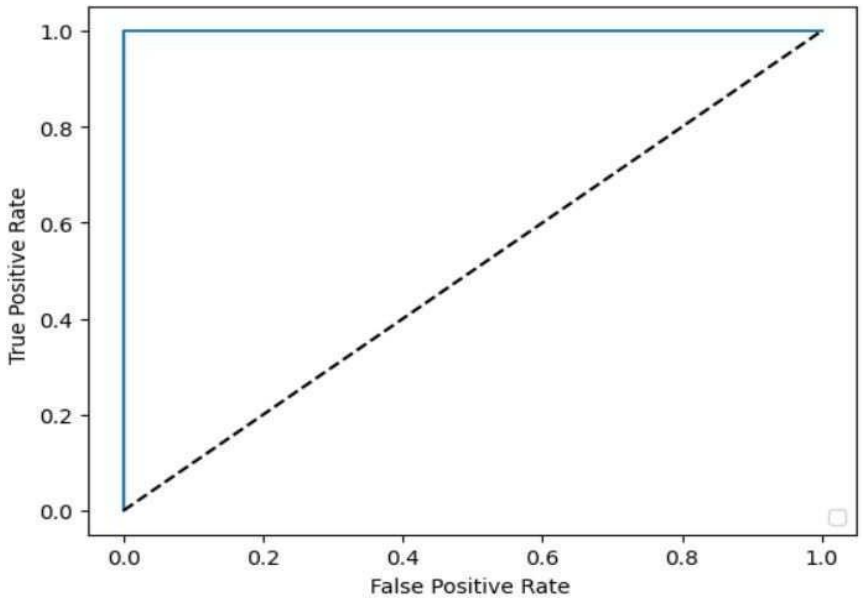


Figure 30 Clear ROC Curve for the comparative scheme

```
print('roc_auc_score: ', roc_auc_score(y_test, y_pred_proba))
```

```
roc_auc_score : 0.84635392938937
```

Figure 31 AUC Score for the comparative scheme

ROC results for our comparative scheme is shown on Figure 30 and Figure 31. When comparing the two ROC curves, the comparative curve is outperformed by our proposed scheme curve. Our proposed scheme curve has a higher AUC value, indicating robust overall performance and a superior ability to classify instances correctly.

How to obtain the best *k* value for the KNN machine learning algorithm using the Minimum error method and RandomisedSearchCV: Error Rate for *k* values

Given that KNN relies on the selection of ***k*** values to determine the number of neighbours considered for classifying a query point, determining the appropriate ***k***

value for training and testing the model can be challenging. As shown above, we tried different ***k*** values before we could get the one suitable for our data.

Therefore, in this study, we provide a quick way of finding the best k value with minimum error as shown in Figure 32. In the below calculation, the value of i can range from 1 to any number, depending on how big the dataset is for training. Then we fit the KNN algorithm with every i from the chosen range. Error is the average of all misclassified observations. From the results, we can now know which value of k gives the minimum error and use that value of k.

```
error = []

#calculating error for k values between 1 to 20
for i in range(1, 20):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, Y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != Y_test))
```

Figure 32 Calculating error for k values

The ***k*** value is an important parameter in K-Nearest Neighbour (KNN) algorithm as it determines the number of neighbouring data points used to classify new data points. In this study, we evaluated the impact of different ***k*** values on the accuracy of the KNN algorithm. Through our analysis (From Figure 27 and Figure 27 below), we found that the most suitable ***k*** value to use is 1, as it produced the least amount of error in classification.

We also observed that as we increased the value of ***k***, the accuracy of the KNN algorithm decreased, and more observations were misclassified. This decrease in accuracy with the increase in ***k*** can be attributed to the smoothing effect that results from considering a larger number of neighbours. As more neighbours are considered, the decision boundary becomes less distinct, leading to the misclassification of data points that lie near the boundary. Our study demonstrates the importance of selecting

an appropriate ***k*** value when using KNN for classification tasks. Choosing the optimal ***k*** value is a crucial step in achieving high accuracy in KNN classification.

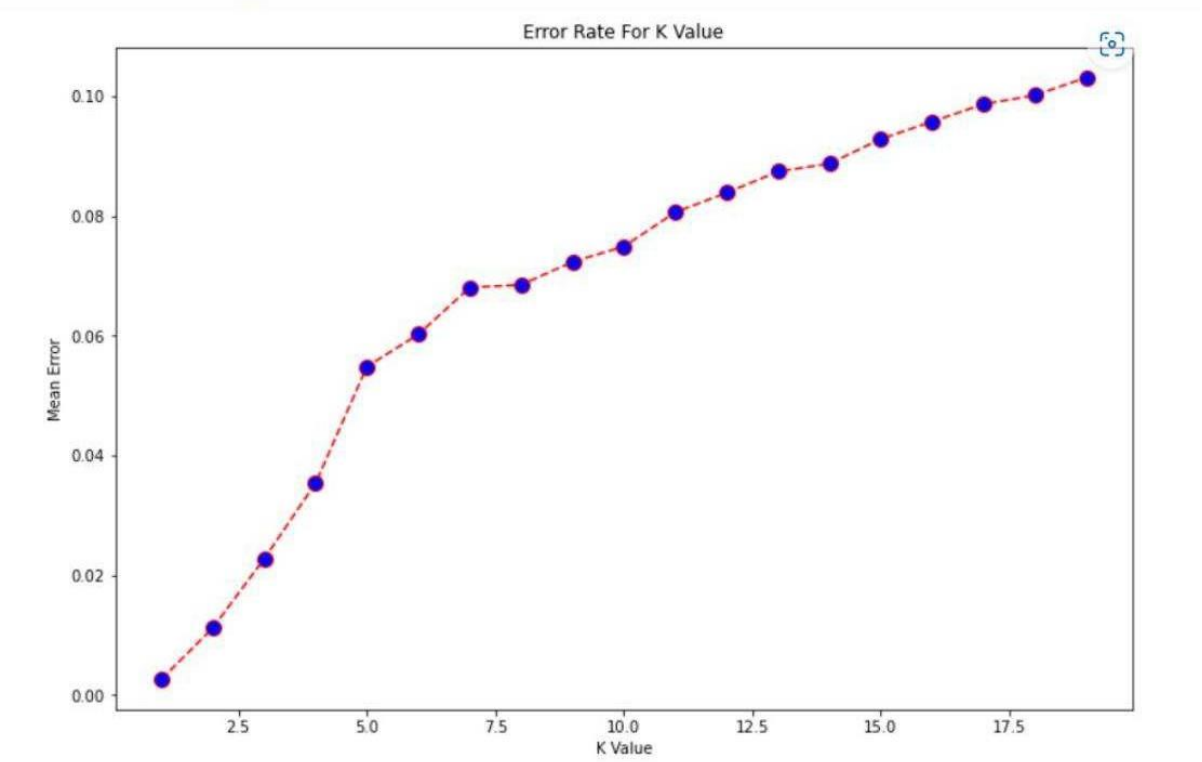


*Figure 33 Error Rate for **k** Values*

```
from sklearn.model_selection import train_test_split, GridSearchCV,RandomizedSearchCV

k=np.random.randint(1,5,10)

params = {'n_neighbors' : k}

random_search = RandomizedSearchCV(classifier,params,n_iter=5 ,cv=5, n_jobs=-1,verbose=0)
random_search.fit(X_train, Y_train)

print("train score: " + str(random_search.score(X_train, Y_train)))
print("test score: " + str(random_search.score(X_test, Y_test)))
```

```
train score: 1.0
test score: 0.9973714285714286
```

```
print(random_search.best_params_)
```

```
{'n_neighbors': 1}
```

*Figure 34 **k** values using randomized Search CV*

The manual way of finding *k*-values vs. using the Minimum error method and RandomisedSearchCV

Finding the optimal ***k*** value for classification purposes can be a time-consuming task. However, employing the minimum error method and randomisedSearchCV technique significantly accelerates this process compared to the traditional approach of attempting various ***k*** values and observing their corresponding classification accuracy. Utilising the minimum error method and randomisedSearchCV, the search for an appropriate ***k*** value for our specific dataset merely required a mere 1 minute, whereas employing the manual method necessitated 2 minutes for each attempt, resulting in a cumulative duration of 10 minutes.

By leveraging the minimum error method and randomisedSearchCV together, the search for an optimal ***k*** value becomes significantly faster and less cumbersome. These methods streamline the process by automatically evaluating a subset of ***k*** values and quickly identifying the one that leads to the most accurate classification results. Consequently, this approach saves valuable time and computational resources, allowing for more efficient model optimisation.

Figure 35 below shows the time taken to find ***k*** values using our dataset.

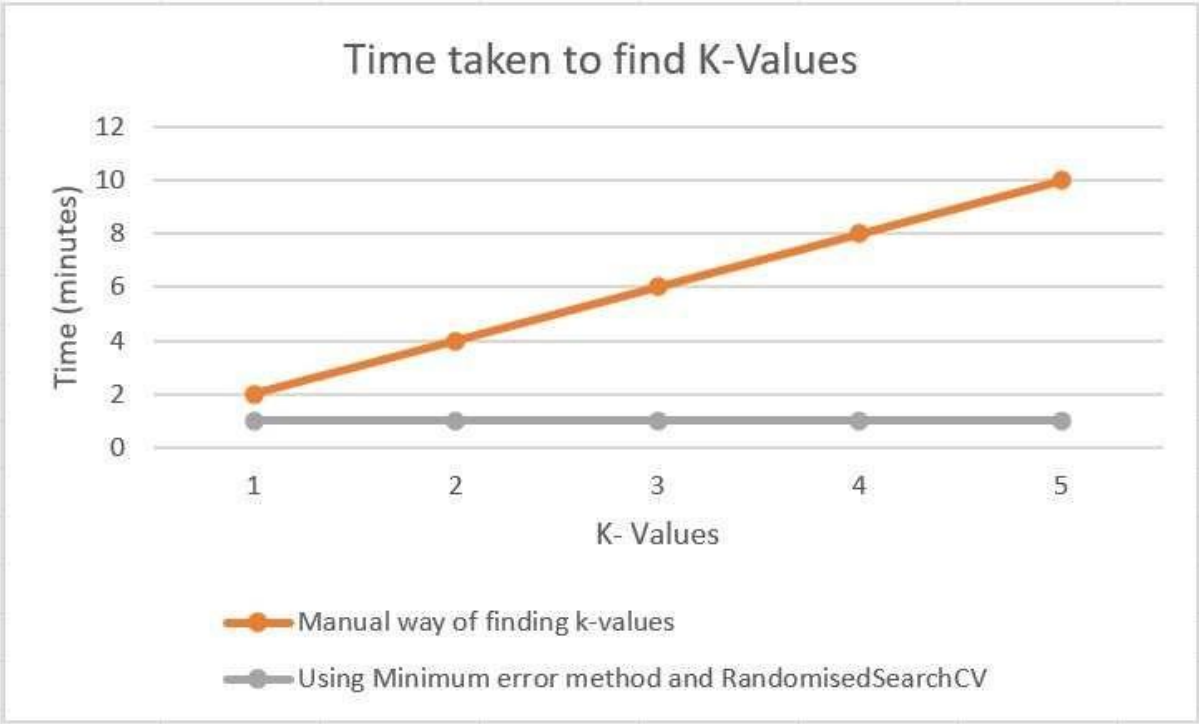


Figure 35 Manual way of finding K-values vs. using Minimum error method and RandomisedSearchCV.

DISCUSSION

In this study, we trained and tested two models, Random Forest (RF) and K-Nearest Neighbours model (KNN). We worked on improving KNN since RF resulted in 100% accuracy.

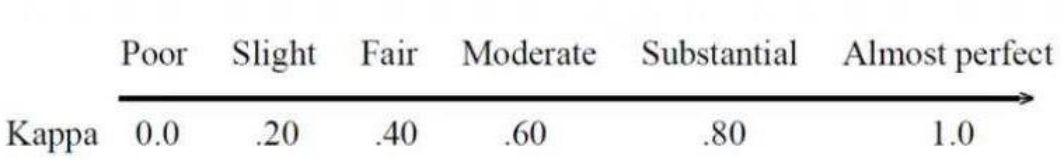
We used accuracy, sensitivity, and specificity (True Negative rate), Cohen's kappa and confusion matrix to analyse the performance of the KNN model.

KNN with n_neighbours = 5 gives 95% accuracy, which we worked on improving in this study and showed the final results in Figure 23. This study shows that KNN prediction improves (gives much better accuracy) when we decrease n_neighbours. Figure 23 shows that KNN decreases accuracy whenever we increase n_neighbours. When n_neighbours = 2, KNN results in 99% accuracy. Therefore, we can conclude that using n_neighbours = 1 results in KNN predicting all values with 100% accuracy and that detecting DDoS using our findings on KNN can help better the model's accuracy.

When improving KNN (using n_neighbours less than 5) we achieve sensitivity and specificity that falls within the range of 95%- 98% from which we can then conclude that our model is significantly superior and excellent.

Cohen's Kappa is a test used to measure the agreement between two raters. In Cohen's Kappa, the degree of agreement between two raters is calculated on a scale of 0 to 1, with 1 being perfect agreement and 0 being no agreement at all. The higher the Cohen's Kappa score, the greater agreement between the two raters. In this study, we used Cohen's Kappa to analyse the performance of our model.

The following picture represents the interpretation of Cohen Kappa:



Kappa	Agreement
< 0	Less than chance agreement
0.01–0.20	Slight agreement
0.21– 0.40	Fair agreement
0.41–0.60	Moderate agreement
0.61–0.80	Substantial agreement
0.81–0.99	Almost perfect agreement

Figure 36 Interpretation of Cohen Kappa Score

Cohen's kappa of our models is 0.98 which falls within the range of 0.81- 0.99. As indicated in Table 3.1, our model is rated as good. The accuracy, precision, sensitivity, and specificity scores fall in the range of 0.8-1 and according to Figure 25, our model is rated as excellent.

CONCLUSION

In conclusion, the results of our study show that the improved K-NN algorithm with a n_neighbours value of 2 has a higher accuracy rate of 98.88% compared to the Decision Tree classifier, which had an accuracy rate of 95%.

The results of this study provide compelling evidence that improving the K-NN classifier can indeed lead to a substantial improvement in the detection and mitigation

of Distributed Denial of Service (DDoS) attacks within Software-Defined Networks (SDNs). Through experimentation and fine-tuning of various parameters, we observed notable enhancements in accuracy and responsiveness. These findings hold significant implications for the field of network security, as a more robust and efficient DDoS detection system can strengthen the resilience of SDNs against malicious threats. Moreover, the research contributes valuable insights into the optimization of machine learning models in practical network security applications, highlighting the potential for further advancements in the ever-evolving landscape of cybersecurity.

Based on these findings, we can confidently conclude that KNN is a more suitable classification algorithm for the dataset under consideration. This study highlights the importance of selecting the appropriate classification algorithm based on the specific problem and dataset, as the accuracy of the model can significantly impact its overall performance. Further research is recommended to explore other factors that may impact the performance of these algorithms and to validate these results on other datasets.

CHAPTER 5: CONCLUSION AND FUTURE WORK

INTRODUCTION

In this chapter, we provide a conclusion to the research presented in this thesis, followed by recommendations for future research work in this area. This study aimed to investigate the detection and mitigation of DDoS attacks in the SDN environment. We presented a comprehensive literature review, followed by the proposed DDoS detection and mitigation framework. We then evaluated the model's performance against various types of DDoS attacks.

Summary on how the objectives where achieved:

We commenced by conducting a thorough literature review to understand the existing knowledge and research related to DDoS attacks in SDN. Familiarizing ourselves with the key concepts, technologies, and security measures in SDN.

We followed these steps to achieve the objectives:

1. Problem Definition: Clearly defining the classification problem to be addressed.
2. Data Collection and Preprocessing: Gather relevant data for your classification task.
3. Data Splitting: Split the dataset into training, validation, and test sets.
4. Feature Selection/Extraction: Identify relevant features and perform feature selection or extraction if needed to reduce dimensionality and improve model performance.
5. Classifier Implementation: Implement Random Forest and KNN classifiers using appropriate libraries or frameworks.
6. Model Training: Train both classifiers using the training data.
7. Validation: Evaluate the classifiers on the validation set to choose the best-performing model. Metrics like accuracy, precision, recall, F1-score, and ROC-AUC.
8. Model Comparison: Compare the performance of Random Forest and KNN classifiers with that of the Decision Tree classifier.
9. Testing: Evaluate the selected models.
10. Results Analysis: Analyze the results obtained from testing.

11. Visualization: Visualize the classification results using appropriate plots (e.g., confusion matrices, ROC curves) to provide a clear understanding of model performance.
12. Conclusion: Summarize findings regarding the effectiveness of Random Forest and KNN classifiers for this work. Discuss which model performed better and why.
13. Report and Documentation: Create a detailed report or documentation of the investigation, including methodology, results, visualizations, and conclusions.

RECOMMENDATIONS

While the proposed framework is effective in detecting DDoS attacks, there is still work that could be further explored in this area. Below are some potential areas for future research:

1. Multi-domain mitigation: This entails expanding the proposed framework to include multiple SDN domains to provide complete protection against coordinated DDoS attacks.
2. Real-world validation: This entails evaluating the proposed framework in a real-world SDN environment to ensure its reliability and scalability.

In summary, the research presented in this study provides a foundation for future research in the area of DDoS mitigation in the SDN environment. With the continued growth of SDN and the increasing frequency and severity of DDoS attacks, research in this field could practically ensure the security and reliability of our networks.

FINAL CONCLUSION

Detecting and mitigating DDoS Attacks is a complex task. At the beginning of this research, we aimed to detect and mitigate DDoS attacks from SDN. We used the downloaded data which was generated from SDN. We cleaned the data and prepared it for model training. We, therefore, trained the data using KNN and RF.

After training both KNN and RF, we found that it is still not entirely satisfactory. Hence, to improve the detection rate further, we proposed using k-neighbours of 5 when training KNN. Our approach is based on KNN (n=5) algorithm with weighted voting.

In this study, we did not integrate RF and K-NN classifiers for several compelling reasons. Firstly, both RF and KNN are standalone machine learning algorithms, each with its own strengths and characteristics. Trying to integrate them into a single model led to complexity and processing overhead. Secondly, RF is an ensemble method that operates by aggregating the predictions of multiple decision trees, while KNN relies on instance-based learning. Combining these distinct approaches in a coherent and effective manner presents considerable challenges, including coordinating different decision-making processes and handling varying data requirements.

Our results show that the improvement of K-NN has the best performance among all the approaches considered in this thesis.

REFERENCES

[1] S. B. M. C. V. Ali Amer, “A survey of DDoS Attacks and Defense Mechanisms in SDN,” 2017.

[2] B. B. N. D. S. Z. Mohammad Shojafar, DDoS Attack Detection and Mitigation in Software Defined Networking, 2016.

[3] S. T. B. S. K. Moiz Ahmed, Interviewee, *A survey of DDoS Defense Mechanisms in SDN.* [Interview]. 2018.

[4] “Kaggle,” [Online]. Available: https://www.kaggle.com/code/favadhassanjaskani/saqib.

[5] D. M. J. Rukshan, “Machine Learning And Statistical Methods for DDoS Attack Detection And Defense System In Software Defined Networks,” Ryerson University, Toronto, Ontario, Canada, 2018.

[6] H. Polat, O. Polat and A. Cetin, “Detecting DDoS Attacks in Software-Defined Networks Through Feature Seletion Methods and Machine Learning Models,” pp. 1-16, 1 February 2020.

[7] H. H. A. A. S.H Ahmed, “DDoS Network in the Era of Cloud Computing and Software-Defined Network,” in *IEEE Network*, 2016.

[8] A. S. Anjana Goel, DDoS Attacks and Defense Mechanisms: A Review, 2016.

[9] M. Y. a. M. R. M. M. Shirali-Shahreza, SDN-Based DDoS Attack Detection and Mitigation: A Survey, IEEE Communications Surveys & Tutorials, 2016.

[10] S. E. Mahmoud, L.-K. Nhien-An, D. Soumyabrata and D. J. Anca, “Machine-Learning Technology for Detecting Attacks in SDN,” ADAPT SFI Research Centre, Dublin, Ireland, 2 Oct 2019.

[11] P. Kaur, M. Kumar and A. Bhandari, “A review of detection approaches for distributed denial of service attacks,” *Systems Science & Control Engineering,* vol. 5, no. 1, p. 301–320, 2017.

[12] S. D. Kotey, E. T. Tchao and J. D. Gadze, “www.mdpi.com/journal/technologies,” 2019.

[13] J. Pei, Y. Chen and W. Ji, “A DDoS Attack Detection Method Based on Machine Learning,” in *IOP Conf. Series: Journal of Physics: Conf. Series 1237 (2019) 032040*, Beijing (100124), China, 2019.

[14] Thet, K. P. T. Htun and Kyaw, “Detection Model for Denial-of-Service Attacks using Random Forest and K-Nearest Neighbors,” *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET),* vol. 2, no. 5, pp. 1855-1859, May 2013.

[15] K. Sukhveer, S. Japinder and S. G. Navtej, “Network Programmability Using Pox Controller,” in *International Conference on Communication, Computing & Systems*, Ferozepur, Punjab, India, August 2014.

[16] F. Pedregosa, A. Gramfort, G. Varoquaux and V. Michel, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research,* January 2012 2012.

[17] W. McKinney, “Pandas: a Foundametal Python Library for Data Analysis and statistics,” January 2011.

[18] M. Mousa, A. Bahaa-Eldin and M. Sobh, “Software Defined Networking Concepts and Challenges,” in *Research gate*, December 2016.

[19] M. Hussain, . N. Shah and . A. Tahir , “Graph-Based Policy Change Detection and Implementation in SDN,” 8 October 2019.

[20] P. Ranjan, P. Pande, R. Oswal, Z. Qurani and R. Bedi, “A Survey of Past, Present and Future of Software Defined Networking,” *International Journal of Advance Research in Computer Science and Management Studies,* vol. 2, no. 4, pp. 238-246, 2014.

[21] M. Dhawan, R. Poddar, K. Mahajan and V. Mann, “SPHINX: Detecting Security Attacks in Software-Defined Networks”.

[22] A. Z. Mahmood , A.-a. A. Nasir and W. H. Fatima , “Performance Evaluation and Comparison of SoftwareDefined Networks Controllers,” *International Journal of ScientificEngineering and Science,* vol. 2, no. 11, pp. 45-50, 2018.

[23] S. Asadollahi and B. Goswami, “Experimenting with Scalability of Floodlight Controller in Software Defined Networks,” in *2017 International Conference on Electrical, Electronics, Communication, Computer and Optimization Techniques (ICEECCOT)*, India, 2017.

[24] S. Streit and C. Kalialakis, “A POX OpenFlow Loop Solution for Mininet Network Emulations,” in *4th International conference on Modern Circuits and System Technologies*, Greece, 2015.

[25] N. Tripathi and B. Mehtre, “DoS andDDoSAttacks:Impact, Analysis and Countermeasures,” in *Researchgate.net*, December 2013.

[26] K. R. Bandara, T. S. Abeysinghe, A. J. Hijaz, D. G. Darshana, H. Aneez, S. J. Kaluarachchi, K. V. Sulochana and D. Dhammearatchi, “Preventing DDoS

attack using Data mining Algorithms," *International Journal of Scientific and Research Publications ,* vol. 6, no. 10, pp. 390-398, 2016.

[27] P. Mahesh, "Random Forest classifier for remote sensing classification," *International Journal of Remote Sensing,* vol. 26, pp. 217-222, 2005.

[28] A. Cutler, D. R. Cutler and J. R. Stevens, "Random Forest," in *Machine Learning*, researchgate.net, 2011, pp. 1-8.

[29] G. Louppe, "UNDERSTANDING RANDOM FORESTS FROM THEORY TO PRACTICAL," University of Liège , 2015.

[30] R. . L. Santos de Oliveira , A. . A. Shinoda , C. . M. Schweitzer and L. R. Prete, "Using Mininet for emulation and prototyping Software-defined Networks," in *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, Colombia, 2014.

[31] C. Comte, "Manipulating and analyzing data with pandas," France, 2019.

[32] C. E. Metz, "Basic principles of ROC analysis," pp. 2383-298, 1978.

[33] T. Fawcett, "An introduction to ROC analysis in pattern recognition letters," pp. 861-874, 2006.

[34] J. S. N. S. G. Sukhveer Kaur, "Network Programmability Using POX Controller," in *International Conference on Communication, Computing & Systems*, August 2014.

[35] A. A. Ominike , A. O. Adebayo and F. Y. Osisanwo, "Introduction to Software Defined Networks (SDN)," *International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868,* vol. 11, no. 7, pp. 1-6, December 2016.

[36] L. Ming-Yi and Y. Chu-Sing, "Design and evaluation of deep packet inspection system: case study," March 2012.

[37] C. Fuch, "Implications of Deep Packet Inspection (DPI) Internet Surveillance for Society," no. 1, pp. 1-5, July 2012.

[38] M. Mohammed, B. K. Muhammad and E. B. M. Bashier, Machine Learning Algorithms and Applications, London, New York: CRC Press, July 2016.

[39] S. Weston and R. Bjornson, "Introduction to Anaconda," April 2016.