Republic of Iraq
Ministry of Higher Education and Scientific Research
University of Babylon
College of Information Technology
Department of Information Networks

A DYNAMIC ALLOCATION AND MACHINE LEARNING SYSTEM TO IMPROVE NEXT-GENERATION NETWORKS SECURITY AGAINST DDOS ATTACKS

A Dissertation

Submitted to the Council of the College of Information Technology, University of Babylon in Partial Fulfillment of the Requirements for the Degree of Doctorate of Philosophy in Information Technology-Information Networks

BY

MOHAMMAD JAWAD KADHIM ABOOD

Supervised by

Prof. Dr. Ghassan Hameed Abdul-Majeed

2023A.D. **1445 A.H.**

بسم الله الرحمن الرحيم

(يَرْفَعِ اللَّهُ الَّذِينَ آمَنُوا مِنكُمْ وَالَّذِينَ أُوتُوا الْعِلْمَ دَرَجَاتٍ)

صدق الله العلي العظيم

سورة المجادلة الاية (١١)

Supervisor Certification

I certify that the dissertation entitled (**A Dynamic Allocation and Machine Learning System to Improve Next-Generation Networks Security Against DDoS Attacks)** was prepared under my supervision at the department of Information Networks/ College of Information Technology/ University of Babylon as partial fulfillment of the requirements of the degree of Doctorate of Philosophy in Information Technology-Information Networks.

Signature:

Supervisor Name: Prof. Dr. Ghassan Hameed Abdul-Majeed

Date: / /2023

Head of the Department Certification

In view of the available recommendations, I forward the dissertation entitled "**A Dynamic Allocation and Machine Learning System to Improve Next-Generation Networks Security Against DDoS Attacks**" for debate by the examination committee.

Signature:

Assist. Prof. Dr. Alharith Abdulkareem Abdullah

Head of Information Networks Department

Date: / /2023

Certification of the Examination Committee

We hereby certify that we have studied the dissertation entitled (**A Dynamic Allocation and Machine Learning System to Improve Next-Generation Networks Security Against DDoS Attacks**) presented by the student (**Mohammad Jawad Kadhim**) and examined him in its content and what is related to it, and that, in our opinion, it is adequate with (**Viva Result**) standing as a dissertation for the degree of Doctor of Philosophy in Information Technology-Information Networks.

Signature:
Name: Dr. Wesam S. Bhaya
Title: Professor
Date: / / 2023
(Chairman)

Signature:
Name: Dr. Rafah Mohammed Kadhim
Title: Professor
Date: / / 2023
(Member)

Signature:
Name: Dr. Mahmood Zaki Abdullah
Title: Assistant Professor
Date: / / 2023
(Member)

Signature:
Name: Dr. Ameer K. Hadi
Title: Assistant Professor

Date: / / 2023
(Member)

Signature:
Name: Dr. Firas Sabah Al-Turaihi
Title: Assistant Professor
Date: / / 2023
(Member)

Signature:
Name: Dr. Ghassan H. Abdul-Majeed
Title: Professor
Date: / / 2023
(Member and Supervisor)

Approved by the Dean of the College of Information Technology, University of Babylon.

Signature:
Name: Dr. Wesam S. Bhaya
Title: Professor
Date: / / 2023
(Dean of Collage of Information Technology)

Dedication

To the souls of those who protected our country and saved the people.

To those who sacrificed themselves to let others stay alive with dignity.

To the souls of the martyrs of the popular crowd and the security forces,

I dedicate this humble work

Acknowledgement

First of all, All Praise be to Allah, the Cherisher and Sustainer of the worlds; for guiding me during my Ph.D. journey. Then, I would like to express my most sincere appreciation for my father and mother for all their sacrifices that have made me what I am today, who strives to seek their happiness and satisfaction. Thanks to my wife, daughters, brothers, and sisters for their continuous support and encouragement to finish this work.

I wish to express my sincere gratitude and appreciation to my supervisor, Dr. Ghassan Hameed Abdul-Majeed for his outstanding supervision, also I want to gratitude and appreciate all my teachers for their valuable advice, and fruitful discussions throughout my Ph.D. journey especially Dr. Wessam Samer, Dr. Eman Salih, Dr. Ghaidaa Al-Mulla and Dr. Mhdi Ebady. Moreover, I am also grateful to my College for giving me the chance to complete my Ph.D. degree.

Last but not least, I would like to thank all my colleagues and staff at the College of Information Technology for making it an enjoyable working environment.

Abstract

Next-generation networks (NGNs) represent the evolution and convergence of various communication technologies, services, and networks to meet the increasing demands of modern society. NGNs aim to provide enhanced capabilities, improved performance, and greater flexibility compared to traditional networks. One of the fundamental features of NGNs is their ability to support multi-access networks, allowing users to connect via different access technologies like wired, wireless, and mobile networks. This enables ubiquitous connectivity and seamless mobility, empowering users to access services anytime, anywhere, and on any device but this feature has a security effect that can reduce the performance of the network and decries the integrity of the transferred data. Distributed Denial of Service DDoS attacks are a significant threat to NGNs. Attackers overwhelm network resources with a massive volume of traffic from multiple sources.

This study proposed a scalable infrastructure that can dynamically allocate resources and adjust traffic flow to mitigate the impact of DDoS attacks. Five different algorithms are used to find the most accurate algorithm and the fastest one regarding training time. In testing, random forest RF has the highest accuracy where J48 represents the fastest algorithm in training time both algorithms are stacked together to get a new stacked model which got 99.9944% accuracy in offline testing time.

The primary objective of the stacked model is to develop a DDoS Classifier using a machine learning algorithm and deploy it within a Docker container in a virtual environment

To efficiently handle potential attacks, an orchestrator was employed to oversee the scaling process, dynamically creating additional instances of the classifier. The number of classifiers instantiated depended on the available resources (CPU, RAM) and the attack intensity. The results indicate that the proposed approach effectively mitigates the DDoS attack with an accuracy between 97% to 95.1% according to the intensity of the attack and the available resources.

Declaration Associated with this Thesis

Some of the works presented in this dissertation have been published or accepted as listed below.

Published Papers:

M. Abood and G. Abdul-Majeed, "Classification of network slicing threats based on slicing enablers: a survey", International Journal of Intelligent Networks, vol. 4, p. 103-112, 2023. https://doi.org/10.1016/j.ijin.2023.04.002

SJR (Q2) CiteScore (7.9)

Accepted Papers:

M. Abood and G. Abdul-Majeed, "Enhancing Multi-class DDoS Attack Classification Using Machine Learning Technique" Submitted to Journal of Advanced Research in Applied Sciences and Engineering Technology

Papers Under review:

1. M. Abood and G. Abdul-Majeed, "A Developed Machine Learning and Dynamic Allocation Approach to Improve Next Generation Networks Security" Submitted to Network Security

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Abbreviation	Description
ADASYN	adaptive synthetic sampling
ANN	Artificial neural network
ANTE	Algorithm for the Number of Terminal Extremities
AWS	Amazon Web Services
BCP	Business Continuity Planning
BI	Business Intelligence
BILSTM	Bidirectional Long Short-Term Memory
BNB	Bernoulli Naive Bayes
CAV	Cooperative Adaptive Cruise Control
CIC	Computer and Information Center
CNN	Convolutional Neural Network
CNS	Central Nervous System
CSE	Computer Science and Engineering
CSF	Critical Success Factor
CSNT	Cloud Storage Network Topology
DAC	Digital-to-Analog Converter
DASA	Data Augmentation for Segmentation Accuracy
DFS	Distributed File System
DL	Deep Learning
DNN	Deep Neural Network
DNS	Domain Name System
DOS	Denial of Service
DT	Decision Tree
EEI	Effective Energy Index
ELM	Extreme Learning Machine
FALL	Formal Approaches to Language Learning
FL	Federated Learning
GB	Gradient Boosting
GNB	Gaussian Naive Bayes
IDS	Intrusion Detection System
IMT	Intelligent Microgrid Technologies
IP	Internet Protocol
IPS	Intrusion Prevention Systems
ISP	Internet Service Provider

KDD	Knowledge Discovery in Databases
KNN	K-Nearest Neighbor
LDAP	Lightweight Directory Access Protocol
LO	Leave-One-Out
LR	Logistic Regression
LSTM	Long Short-Term Memory
LSVM	Least Squares Support Vector Machine
LTE	Long-Term Evolution
LSVM	Linear Support Vector Machine
MANO	management and orchestration
MCOM	Multimedia Communications
MEC	Mobile Edge Computing
MI	Mutual Information
ML	Machine Learning
MLP	Multilayer Perceptron
MNB	Multinomial Naive Bayes
MONS	Management of Networks and Services
MP	MultilayerPerseptron,
MSSQL	Microsoft SQL Server
NB	Naïve Bayes
NF	Network Function
NFV	Network Functions Virtualization
NGNs	Next Generation Networks
NO	Numerical Optimization
NS	Network Security
NSL	Network Security Lab
NSM	Network and Systems Management
ONOS	Open Network Operating System
PART	Partial Decision Tree
PP	Privacy Preserving
RBF	Radial Basis Function
RCTFM	Regression Coefficients of Traffic Flow Metric
RF	Random Forest
RFFI	Random Forest Feature Importance
RNN	Recurrent Neural Network
ROS	Robot Operating System
RUB	Ruhr University Bochum

SDK	Software Development Kit
SDN	Software-Defined Networking
SEC	Security
SLO	Service Level Objective
SMOTE	Synthetic Minority Over-sampling Technique
SN	Sensor Network
SOC	Security Operations Center
SOCC	Service-Oriented Computing and Cloud Computing
SVM	Support Vector Machine
SYM	Symposium
SYN	Synchronize
VF	Virtual Function
VG	Variational Gaussian
VM	virtual machines
VNF	Virtualized Network Function
VS	Virtualization Server
WM	Wearable Monitor
WVE	Wireless Virtualization Environment
XEN	Xen Project is an open-source hypervisor used for virtualization
YSA	Yield-Sensitive Approximation

CHAPTER One

INTRODUCTION

1.1 Overview

The exponential growth of smart devices and their services, applications, and traffic data, in addition to the rapid development of the Internet of Things (IoT), has triggered global initiatives toward developing the next generation of communication systems [1]–[4]. The rapidly growing number of network-connected devices and the increasing of bandwidth-hungry applications require higher spectral efficiency than the old-generation systems have. In Table (1.1), Cisco Annual Internet Report (2018–2023) White Paper [5] mentions the global internet adoption, devices, and connection.

Table 1-1 Global Internet Adoption, devices, and Connection [5]

Metric	2018	2023
Internet users	3.9 billion	5.3 billion
number of IP devices	2.4 billion per capita	3.6 billion per capita
Machine to Machine connections	33 percent	50 percent
global mobile subscribers	5.1 billion	5.7 billion

The numbers in Table (1-1) mean that the next-generation networks need to deliver capacity more than 1000 times compared to the current commercial 4G cellular systems [3], [6].

Next Generation Networks (NGNs) have emerged as a revolutionary communication infrastructure, designed to address the escalating demands of modern connectivity. It represents the next phase of network architecture which provides a robust and scalable foundation for diverse communication services and surpasses the limitations of traditional networks. This technology enables seamless integration of voice, data, and multimedia applications over a unified network, fostering flexibility and efficiency.

One of the key advantages of NGNs is the convergence of multiple communication services onto a single platform. By eliminating the need for separate infrastructures, these networks streamline communication processes, reduce complexity, enhance user experience, and converged services including voice calls, video conferencing, messaging, data transfer, and multimedia content delivery. Also, it offers unparalleled scalability and flexibility, allowing for easy expansion and adaptation to changing requirements. As network traffic and user demands grow, NGNs can seamlessly accommodate the increased load by scaling resources, adding new services, integrating emerging technologies, and prioritizing QoS by leveraging advanced mechanisms to ensure optimal performance for real-time applications [7]. With features like traffic shaping, packet prioritization, and bandwidth allocation, NGNs can provide improved call quality, reduced latency, and seamless multimedia experiences. In addition, NGNs provide a robust platform for deploying advanced applications and services. Cloud computing, Internet of Things (IoT), virtualization, vehicle to everything (V2X), and multimedia applications find a conducive environment within NGNs [8].

NGNs bring significant advancements to communication infrastructure, but it also introduce new challenges, particularly in terms of security. DDoS attacks pose a severe threat to NGNs, potentially disrupting services and causing financial losses. The proliferation of NGNs has brought numerous benefits to the communication landscape. However, these advanced networks also face an increased risk of DDoS attacks. with their increased bandwidth capabilities, may become attractive targets for attackers looking to exploit this scalability. The intensity of these attacks can overwhelm network devices, exhaust network resources, and impact legitimate traffic.

Also, Attackers can exploit vulnerabilities in the network components, protocols, or applications to launch sophisticated attacks that evade detection and mitigation mechanisms. The influx of malicious traffic during a DDoS attack can significantly increase network latency. Legitimate user requests may experience delays in reaching their destination, leading to slower response times and degraded performance for network services. In some cases, DDoS attacks against specific services or nodes on an NGN can cause collateral damage to other connected services or infrastructure. For example, an attack targeting a specific website hosted on an NGN can cause network disruptions for other websites or services sharing the same infrastructure.

In this dissertation, the proposed work is developing a flexible elastic defense system capable of absorbing DDoS attacks by designing ML classifiers that deploy as microservices inside a network slice. The defensive system has the ability to scale up or down regarding the available resources and the intensity of the attack to reduce the latency and decrease the amount of delay resulting from the increase in requests.

Kubernetes is used in addition to the ML algorithm to orchestrate the scaling process and to improve the performance of the defense system, which proved to be more efficient than the systems in which Software-Defined Network SDN was used, because the systems that used SDN suffered from major problems represented in the difficulty of managing resources, determining the number of Virtual Machine (VM) sand their locations and because they depend on A per-flow synchronization model, where switches need to contact the network controller every time they receive a new flow.

The algorithm used to accomplish the classifier was chosen based on a set of experiments in which a different set of algorithms were used, and the best one was chosen among them. The attack was implemented with different scenarios by using a group of physical and VMs. The evaluation shows an effective response in dealing with attacks, in addition, the system has proven an ability to scale up or down in a way that ensures the preservation and use of resources optimally.

1.2 Related Works

Many research papers used machine learning algorithms to mitigate DDoS attacks, only those that are relevant to the proposed study are included. References [9]-[17] are related to the ML side of the study and summarized in Table 1.2.

Elsayed, et al., (2020) [9] Developed a DDoSNet Intrusion Detection System for environments controlled by SDN. The proposed system was based on DL and combines RNN and autoencoders. The developed system was evaluated using the CIC-DDoS2019 [10] dataset. The authors exhibited a significant improvement in assault detection over previous methods. As a result, the suggested strategy provides high confidence in SDN environment preservation.

Bolodurina, et al., (2020) [11] Used the CIC-DDoS2019 dataset, which contains reflection-based and exploitation-based attack information, to investigate the effect of a data balancing technique on the network traffic categorization problem on various types of DDoS attacks. In categorizing network risks, the results indicated the efficacy of data balancing methodologies such as synthetic minority sampling, naive random, and adaptive synthetic sampling.

Aytac, et al., (2020) [12] employed a variety of ML approaches, to assess the success rate of the intrusion detection system. The CIC-DDoS2019 dataset was used to study many ML models, including the Artificial neural network (ANN), Support Vector Machine (SVM), Gaussian Naive Bayes (GNB), Multinomial Naive Bayes (MNB), Logistic Regression (LR), K-Nearest Neighbor (KNN), Decision Tree (DT), and Random Forest (RF) methods. The authors demonstrated that the best predictions are made using KNN, LR, and NB.

Cil, et al., (2020) [13] Proposed an ML method to identify DDoS attacks on a network traffic packet sample, using a Deep Neural Network (DNN) as a deep learning technique. Because it includes feature extraction and classification algorithms, the DNN model may execute rapidly and accurately even with small sample sets. The authors used the CIC-DDoS2019 dataset, which covers a variety of DDoS attack types developed in 2019. The proposed method achieves an accuracy of 94.57%.

Kushwan and Ranga, (2021) [14] proposed a hybrid DDoS detection method based on ML. The suggested system is made up of the Extreme Learning Machine (ELM) algorithm and the black-hole optimization technique. The authors conducted multiple tests with various data sets to assess the performance of the proposed hybrid ML system. The proposed hybrid approach detects DDoS assaults in cloud computing with a detection accuracy of 99.8 percent using the CIC-DDoS2019 dataset.

Kousar, et al., (2021) [15] Suggested a detection system based on numerous classification algorithms utilizing the CIC-DDoS2019 dataset, which was capable of detecting various types of DDoS attacks. Furthermore, the authors captured packets from the software development kits (SDK) environment, preprocessed the dataset, and then used a classification method to detect DDoS attacks. The DT outperforms SVM and NB ML models, according to the authors.

Gadallah et al. (2021) [16] proposed an effective and robust method for detecting DDoS attacks by leveraging ML algorithms that utilize advanced features derived from traffic flow data and statistics. The authors train a model using the radial basis function kernel, employing a controller that extracts packet headers and performs flow computations to acquire the necessary characteristics. These features are then combined to create a dataset that is inputted into a Linear Support Vector Machine (LSVM) classifier. The model was trained using the kernel radial basis function to enhance the classifier's performance.

Araujo, et al., (2022) [17] Presented an Algorithm for the Number of Terminal Extremities (ANTE), an ML-based system for ANTicipating botnEt signals. The technology adjusts to various settings and recognizes various sorts of botnets. It automatically finds the best ML pipeline for each botnet and improves categorization before starting the attack. The system evaluation was based on trace-driven experiments that compare the results to other relevant results from the literature across four typical datasets: ISOT HTTP Botnet, CTU-13, CIC-DDoS2019, and BoT-IoT. The results showed that the average detection accuracy is 99.06%. Table 1-2. The ML-related works and the proposed method.

Table 1-2 Summary of The ML-related Works and the proposed method

No	Reference	ML techniques applied	dataset	Result
1	Elsayed et al., (2020) [9]	DT , NB, Booster, RF , SVM, LR and RNN-autoencoder	CIC-DDoS2019	RNN-autoencoder was the better with an accuracy of 99.0%
2	Bolodurin a et al., (2020) [11]	ROS, SMOTE, and ADASYN	CIC-DDoS2019	When compared to previous methods, the ADASYN adaptive synthetic sampling method enhanced attack categorization accuracy by up to 98%.
3	Aytac et al., (2020) [12]	LR, YSA, GNB, MNB, BNB, KNN, DT (entropy-gini), RF, and SVM	CIC-DDoS2019	The highest success rate of datasets was used for training and test obtained by using KNN, LR, NB, (Multinomial - Bernoulli) algorithms with an accuracy of 99.7%
4	Cil et al., (2020) [13]	Three-layer DNN model	CIC-DDoS2019	The three-layer DNN model to achieve 95%
5	Kushwan and Ranga, (2021) [14]	improved version of the self-adaptive evolutionary extreme learning machine called self-adaptive evolutionary extreme learning machine with crossover adaptation is developed	NSL-KDD, ISCX IDS 2012, UNSW-NB15, and CIC-IDS2017 [18]	It achieves the detection accuracy of 73.00%, 98.90%, 89.17%, and 99.99% with NSL-KDD, ISCX IDS 2012, UNSW-NB15, and CIC-IDS2017 datasets, respectively
6	Kousart al., (2021) [15]	DT, SVM, and NB.	CIC-DDoS2019	the DT has a better performance compared to SVM and NB.
7	Gadallah et al., (2021) [16]	NB, KNN, DT, and RF are also utilized and compared with the SVM	The features are used to create a dataset that includes 50,000 records for normal traffic in addition to 50,000 records for attack traffic	SVM gives the best accuracy among the other ML-based technologies used with an accuracy of 99.84%
8	Araujo et al., (2022) [17]	ANTicipating botnEt signals based on ML algorithms.	ISOT HTTP Botnet, CTU-13, CIC-DDoS2019, and BoT-IoT	the efficiency of the ML pipelines selected by ANTE ranged from 79.2% to 96.69%.

Literatures [19] – [25] are used to determine which feature should be selected to get better results. Table 1.3 summarizes the mentioned literatures.

Silipo et al., (2014) [19] examined various commonly used techniques for dimensionality reduction. These techniques included eliminating columns with excessive missing values, removing columns with low variance, reducing highly correlated columns, applying Principal Component Analysis (PCA), exploring RF, Backward Feature Elimination, and Forward Feature Construction. The researchers showcased the implementation of these techniques within the KNIME platform, utilizing dedicated pre-defined nodes or constructing custom sub-workflows within meta-nodes.

Lucky et al., (2020) [20] A lightweight decision tree model, inspired by the C4.5 algorithm, has been introduced by employing a reliable and efficient feature selection technique. Multiple DT models were created with varying numbers of selected features using the Low Variance filter technique. The feature selection process involved tests with different variance threshold values, and a threshold of 0.025 was identified as optimal. By using only three selected features, the design analysis demonstrates minimal CPU load on the detection system while still achieving high accuracy in detecting malicious activities.
Maniriho et al., (2020) [21] proposed IDS methods to evaluate the effectiveness and efficiency of two different approaches: Lazy Instance-Based Learning (a single machine learning approach) and Random Committee (an ensemble approach) for intrusion detection. The performance of these methods was tested using the best feature subset selected from recent network traffic datasets, namely NSL-KDD and UNSW-NB15. The results

indicated that the ensemble technique outperforms the single machine learning approach, with a misclassification gap of 0.969% and 1.19% (using NSL-KDD), as well as 1.62% and 1.576% (using UNSW-NB15).

Peneti and E (2021) [22] Implemented feature selection methods to develop a highly efficient Intrusion detection system IDS. By reducing features, the speed of the IDS can be improved, and memory usage can be reduced. Additionally, machine learning techniques offer greater efficiency compared to existing IDS approaches. However, some challenges need to be addressed in this system. It should be capable of operating effectively even in adverse scenarios, which necessitates the system being distributed to ensure fault tolerance and resilience.

Zaib et al., (2021) [23] Presented a methodology for the early detection of DoS and DDoS attacks. The problem is formulated in a practical scenario by comparing flow-based and non-flow-based datasets using the Mann-Whitney U statistical test. Classification is performed using ANN and SVM algorithms, with both flow and non-flow-based datasets. To preserve the original features, unimportant features are eliminated using variance, correlation, and the ¾ quartile method. The wrapper method is utilized for feature selection to identify the most relevant features.

Ahsan et al., (2021) [24] Proposed the Dynamic Feature Selector (DFS) in this study aims to enhance prediction accuracy and reduce model complexity by utilizing statistical analysis and feature importance tests. To assess the effectiveness of DFS, experiments were conducted using two commonly used datasets in cybersecurity research: NSL-KDD and the UNSW-NB15 datasets. During the meta-learning stage, four algorithms were compared for accuracy estimation: Bidirectional Long Short-Term Memory, Gated Recurrent Units, RF, and a novel CNN and LSTM (CNN-LSTM) proposed in this study.

Kshirsagar and Kumar (2021) [25] Introduced an efficient framework for detecting reflection and exploitation-based DDoS attacks. The framework is tested on the latest DDoS evaluation dataset CIC-DDoS2019, using the J48 classifier. By employing a feature reduction method, the framework achieves a significant reduction in the number of features, ranging from 56% to 82.92% compared to the original set. Experimental results demonstrate that the proposed framework surpasses the performance achieved by using a subset of reduced features. Furthermore, when validating the framework on the (KDD Cup 1999) dataset, it shows improved performance in both binary and multi-level classification tasks, achieved through a feature reduction of 60.97% from the original feature set.

Table 1-3 Summary of features selection related works and the proposed method

No	Reference	Dataset	Feature selection techniques	Contribution
1	Silipo et al., (2014) [19]	KDD data set	1. Numerous absent data entries 2. Limited variability 3. Strong correlation with other column values 4. PCA (Principal Component Analysis) 5. Initial divisions in decision trees of random forest 6. Step-by-step removal of features 7. Progressive creation of new features	Principal Component Analysis, exploring Random Forests, Backward Feature Elimination, and Forward Feature Construction are the most commonly employed techniques for reducing dimensionality. Additional approaches involve eliminating columns with excessive instances of missing values, decreasing correlated columns, and discarding columns with low variance.
2	Lucky et al., (2020) [20]	CAIDA 2007, CICIDS2017 and CIC-DDoS2019	Low Variance filter technique	Based on the number of features chosen, various DT models were built using the Low Variance filter approach and a variance threshold of 0.025.
3	Maniriho et al., (2020) [21]	CIC-IDS2017 and KDD Cup	Filter-based feature selection algorithms: IGR, CR, and ReF.	With increased performance, the proposed feature reduction method aims to reduce the number of characteristics needed to identify DoS assaults.
4	Peneti and E (2021) [22]	CIC-IDS2017	Recursive Feature Elimination	The aim of this research is to employ feature selection methods for developing a highly efficient intrusion detection system. By removing certain features, the speed of the IDS is enhanced, and memory demands are reduced significantly.
5	Zaib et al., (2021) [23]	CSE-CIC-IDS2018 [10] and NSL-KDD.	The forward selection wrapper	Reduce features using variance, correlation, and ¾ quartile method.
6	Ahsan et al., (2021) [24]	NSL-KDD and UNSW-NB15.	Univariate test and Pearson coefficient test along with XGBoost importance and wrapper technique	XGBoost importance and wrapper approach were used in conjunction with univariate test and Pearson coefficient test statistical analysis as well as various feature engineering processes to reduce feature dimensionality.
7	Kshirsagar and Kumar (2021) [25]	CIC-DDoS2019and KDD Cup 1999	information gain and correlation	The study proposes a DDoS attack detection methodology for quickly detecting refection and exploitation-based DDoS assaults.

Recent studies have suggested that the use of NGN enablers can be an effective tool in mitigating DDoS attacks. Literatures in [26] and [31]-[39] which are summarized in Table 1.4, present the ability to build distributed detection and prevention systems using these tools.

Fayaz et al. (2015) [26] presented the implementation of a flexible and adaptable DDoS defense system called the Bohatei controller. This system is built using the OpenDaylight [27] platform, which is an industry-grade SDN platform widely used in the field. The authors incorporate various open-source tools, including OpenvSwitch [28], Snort [29], Bro [30], and iptables [31], as defense modules within their system. They also develop a resource management algorithm that can scale effectively. The evaluation of Bohatei is conducted both on a real testbed and through simulations.

Fung and McCormick (2015) [32] introduced VGuard, an innovative traffic engineering solution that utilizes prioritization and is built upon a Virtualized Network Function (VNF) for DDoS mitigation. The system directs flows from the external zone to different tunnels based on their priority levels. This approach ensures that trusted and lawful flows receive a guaranteed quality of service, while flows associated with attacks and suspicious activities compete for available resources. The authors present two methods for determining the direction of flows: static and dynamic, providing flexibility in managing traffic within the system.

Jakaria et al. (2016) [33] introduced Vfence, a system that leverages the capabilities of Network Functions Virtualization (NFV) architecture to counter Synchronize (SYN) flood attacks. The NFV architecture enables the implementation of network functions to be flexible and dynamically adaptable. The researchers proposed employing network agents to intercept packets during potential attack scenarios, verifying their legitimacy, and safeguarding the server by discarding suspicious messages.

Rashidi et al. (2017) [34] introduced CoFence, a method for DDoS protection that facilitates network collaboration among NFV-based domain networks through the concept of "domain-helps-domain." CoFence enables domain networks to support each other during intense DDoS attacks by sharing resources. The authors develop a dynamic resource allocation system for domains, ensuring that the distribution of resources is fair, efficient, and incentivizes cooperation among the networks.

Zhou and Guo (2017) [35] proposed a framework for mitigating DDoS attacks that leverages the capabilities of NFV and SDN. The framework utilizes the SDN features of centralized control and a global network view to monitor and analyze network traffic. When anomalous traffic is detected, the framework computes suitable countermeasures and dynamically virtualizes, instantiates, deploys, and connects defense resources to mitigate the attack effectively.

Alharbi, et al., (2017) [36] Proposed a cloud-based DDoS prevention service to enable enterprises to reroute their traffic to the cloud's scrubbing centers for filtering. This method has some flaws, including latency and privacy violations. New networking service models have recently been proposed, including NFV and edge computing. They create a system for DDoS mitigation using two-stage procedures with NFV and edge computing.

Garcia et al. (2018) [37] introduced a hybrid solution called DeMONS to address DDoS attacks. DeMONS comprises five main modules, implemented as VNFs: a priority classifier, firewall, allocation, traffic policing, and manager. Within the DeMONS solution, network flows undergo evaluation by the priority classifier, which assigns them a reputation score ranging from 0 to 1 based on their significance. The allocation module then blocks zero-priority flows at the firewall and allocates the remaining flows into separate tunnels based on their priority (high and low). In cases where the low-priority tunnel becomes overloaded, the Traffic Policing module implements an algorithm to restrict the traffic of each flow based on its priority. The manager module assumes responsibility for provisioning and managing the life cycles of all the modules within the solution.

Mamolar et al. (2018) [38] focused on developing a comprehensive detection system that offers simultaneous protection to infrastructures, tenants, and 5G users in both the edge and core network segments of 5G multitenant infrastructures. they introduced an approach that enhances the capabilities of a commonly used IDS to accurately identify attacking nodes within a 5G network, regardless of the various network traffic encapsulations employed. The proposed method can be deployed across nearly all segments of a 5G network, including Mobile Edge Computing (MEC). The authors also presented the Regression Coefficients of Traffic Flow Metric (RCTFM) as a DDoS technique within this context.

Kalathiripi and Venkatram (2020) [39] employed the RCTFM to perform predictive analysis for detecting the magnitude of DDoS attacks. This analysis is based on transactions that are stored in a buffer over a defined static period. Unlike existing models, the proposed model takes the buffered traffic within the specified timeframe as input and calculates regression coefficients for parameters represented as metrics of the corresponding buffered traffic flow.

Köksal et al. (2021) [40] introduced a novel DDoS mitigation mechanism designed specifically for 5G NFV networks. Their proposed mechanism utilizes VMs as IPS to intercept queries. The management and orchestration (MANO) component dynamically deploys IPS VMs to distribute the load evenly based on the volume of

DDoS traffic. To assess the effectiveness of the mechanism, the researchers conduct experiments within a real 5G NFV environment that is created using appropriate tools specifically designed for 5G NFV environments.

Table 1-4 Summary of relevant studies that used NGNs enablers to mitigate the effects of DDoS attacks

No.	Reference	Technologies	ML	Microservice	Orchestrator	IPS	Main contribution	Experiment effectiveness
1.	Fayaz, et al., (2015) [26]	SDN, NFV,	No	No	Not determined	Not determined	Construct a Flexible and Elastic DDoS Defense System utilizing Open Source SDN and NFV Tools.	It outperforms naive SDN implementations that do not address control plane bottlenecks and respond rapidly to attacks.
2.	Fung and McCormick, (2015) [32]	NFV	No	No	No	Firewall	Prioritizing incoming traffic, introduce a solution for mitigating DDoS attacks using NFV.	Prioritization-based VNF is implemented, and their method can effectively support trusted flows during DDoS attacks.
3.	Jakaria, et al., (2016) [33]	NFV	No	No	MANO	BCP: 38 standard	Utilizing the adaptability of network functions to filter DDoS attacks against online services.	This mitigation technique keeps declining rates of normal traffic at extremely low levels and blocks SYN floods with agents.
4.	Rashidi, et al., (2017) [34]	NFV	No	No	Not used	Set of configurati	Developing a DDoS defense mechanism that relies on domain-to-domain cooperation and solves the resource allocation issue.	Different physical resources of distinct domains are distributed efficiently and equitably to mitigate the effects of DDoS attacks.
5.	Zhou and Guo, (2017) [35]	NFV, SDN	No	NO	Not determine	Not determine	Combining SDN and NFV technologies for DDoS protection.	The mitigation framework is successfully implemented in industrial control systems to mitigate DDoS attacks.
6.	Alharbi, et al., (2017) [36]	NFV, VMs	No	No	MANO	Not determine	NFV and edge computing architectures utilize a two-stage DDoS mitigation framework.	It is anticipated that the proposed mitigation scheme will be implemented on the premises of organizations to increase security and decrease latency.

No.	Reference	Technologies	ML	Microservice	Orchestrator	IPS	Main contribution	Experiment effectiveness
7.	Garcia, et al., (2018) [37]	NFV	No	No	No	Not determined	Build a hybrid DDoS mitigation strategy that employs both capacity and filtering techniques. The solution technique resembles that of vGuard.	Utilizing a reputation system, a policy-shaping module, and an allocation method allowed for improved results in comparison to vGuard.
8.	Mamolar, et al., (2018) [38]	CORE Emulator	No	No	No	Snort	Implement a transversal detection system to safeguard 5G multitenant networks from DDoS attacks.	Experiments conducted in genuine 5G multi-tenant infrastructures validate the scalability and adaptability of the proposed architecture.
9.	Kalathiripi and Venkatram, (2020) [39]	RCTFM	No	No	No	Not determine	Conducting predictive analysis to detect DDoS attacks based on transactions buffered during a specified static time frame.	A comparative experiment demonstrates the efficacy of the proposed model in detecting DDOS attacks over existing methods.
10.	Köksal, et al., (2021) [40]	NFV, VMs	No	No	MANO	Snort	Using MANO to dynamically scale VNFs resources to mitigate DNS deluge attacks in NFV-based 5G networks.	The system's efficacy is evaluated in a real NFV-based 5G network created with OpenStack and Open Source MANO. When the system is entirely operational, the quality of service (QoS) increases, as demonstrated by the results.

In many works of literature, different forecasting techniques have been proposed to enable predictivity approaches. Literatures from [41]–[43] introduces the concept of elasticity in the field of distributed systems to scale up or down the system depending on specific metrics. Table 1.5 summarizes the details of the presented studies.

Shen et al. (2011) [41] introduced CloudScale, a solution designed to automate the precise and flexible scaling of resources in multi-tenant cloud computing infrastructures. CloudScale addresses scaling challenges between applications by utilizing migration techniques. Additionally, it incorporates dynamic CPU voltage/frequency scaling to achieve energy savings without compromising application Service Level Objectives (SLOs). The researchers implemented CloudScale on top of the Xen hypervisor and conducted comprehensive testing using a range of CPU and memory-intensive applications, including RUBiS, Hadoop, and IBM System S. The results demonstrate that CloudScale outperforms other solutions by significantly improving SLO conformance while consuming fewer resources and energy.

Ren, et al., (2016) [42], Proposed a VNF Dynamic Auto Scaling Algorithm (DASA) that balances performance and operation costs. They create an analytical model to quantify the tradeoff and validate it using extensive simulations. Given the latency upper-bound, the results suggest that the DASA can drastically cut operation expenses. Furthermore, the models enable rapid evaluation of the cost-performance tradeoff and system design without widespread implementation, which can save money and time.

Shariffdeen, et al., (2017) [43], suggested a forecasting technique to improve workload forecasting accuracy in cloud auto-scalers. To produce more accurate predictions on significantly varied workload patterns, an ensemble workload prediction mechanism based on time series and ML approaches is proposed. According to simulation findings, our ensemble method yields much fewer forecast errors than individual models and the prediction strategy used in Apache Stratos, an open-source PaaS platform.

Table 1-5 Summary of elasticity research related to this work

No.	Reference	Platform	deployment technologies	Main Contribution	Metrics
1	Shen, et al., (2011) [41]	XEN	VM	CloudScale, an autonomous elastic resource scaling method for multi-tenant cloud computing infrastructures, is demonstrated.	RAM
2	Ren, et al., (2016) [42]	Amazon AWS	VM	Analytical and simulation models were created to investigate the average job response time and operation cost.	Number of waiting requests in the buffer
3	Shariffdeen, et al., (2017) [43]	Apache Stratos	VM	Develop proactive scaling solutions, where future resource demand can be forecasted and necessary scaling steps may be implemented ahead of time, to avoid the danger of under-provisioning at peak loads and over-provisioning at other times.	time series and ML techniques

Finally, the following literatures, [45]–[48] (summarized in Table 1-6) introduces the ability to build network slices with different technologies to implement various implementations.

Khettab et al., (2018) [44] Presented an architecture that will investigate how NFV and SDN can be used to secure a network slice on demand, addressing the new security concerns placed on network management by flexibility and elasticity support. Their proposed approach attempts to achieve optimal resource allocation while managing the slice security policy efficiently. Furthermore, experimental performance evaluations of the security overhead in virtualized contexts are reported.

Sattar and Matrawy (2019) [45] Proposed a mathematical approach to offer on-demand slice isolation while still ensuring end-to-end delay for 5G core network slices. they assess the suggested work using a combination of modeling and experimental studies. their findings indicate that the proposed isolation could help to reduce Distributed Denial-of-Service attacks while also increasing the availability of the slices.

Boualouache and Engelthis (2022) [46] developed a set of security Virtual Network Functions (sVNFs) empowered by deep learning (DL) for V2X-NSs (Vehicle-to-Everything Network Slices). Their implementation combines the concept of Virtual Security as a Service (VSaS) with DL and Federated Learning (FL). The authors introduced a privacy protection scheme that is hierarchical and enables collaborative learning through FL. Their approach also includes an incentive system based on game theory to motivate FL clients (Connected Autonomous Vehicles - CAVs) to contribute high-quality DL local models. The researchers trained, validated, and tested their approach using a publicly available dataset.

Wijethilaka and Liyanage (2022) [47] put forward a security architecture for L5GO (Layer 5G and Beyond) networks utilizing Network Slicing (NS). Their proposed framework aims to enhance the scalability, dynamic nature, and adaptability of the system while also focusing on reducing costs. To validate the functionality of the security framework, the authors conducted experiments on a real testbed. Extensive evaluations were performed to assess the performance of the framework in terms of resource preservation, cost reduction, and variability of latency.

Table 1-6 Summary of network slicing research related to this work

No.	Reference	Implementation environment		Slicing Implementation technology	Deployment Technologies	Using Extended Berkeley Packet Filter?	Main Contribution
		Simulator	Virtual Environment				
1	Khettab et al., (2018) [45]		Yes	SDN, NFV, ONOS, VM	VM	No	Enables Security as a Service (SECaaS) within network slices using SDN and NFV technologies.
2	Sattar and Matrawy (2019) [46]	Yes		NS3	-	No	used slice isolation to reduce the impact of DDoS attacks on a simple network service (slice authentication).
3	Boualouache and Engelthis (2022) [48]		Yes	SDN, NFV, MEC, NSM, SOC	VM	No	Develop a scheme for detecting inter-slice V2X attacks using the flexibility of virtual security as a service concept and the power of deep learning and federated collaborative learning.
4	Wijethilaka and Liyanage (2022) [48]	Yes		MATLAB	-	No	introduced the concept of allocating a dedicated slice of the network to enable the SECaaS paradigm for an NS-enabled L5GO ecosystem

1.3 Problem Statement

In the context of new-generation networks, such as 5G, the threat of DDoS attacks poses a significant challenge to network security and availability. DDoS attacks involve overwhelming a target network or system with a flood of illegitimate traffic from multiple sources, rendering it unable to respond to legitimate requests. Detecting and mitigating these attacks on time is critical to ensure the uninterrupted functioning of new-generation networks.

The problem at hand is to develop a distributed detection and prevention system capable of identifying and mitigating DDoS attacks in next-generation networks. Traditional centralized detection approaches may not be suitable due to the scale and complexity of these networks.

On the other hand, there is a need for technology. which efficiently distributes the network resources to meet the varying demands of users and applications to optimize network performance, enhance user experience, and ensure efficient utilization of network resources.

Moreover, the distributed detection system should be resilient to attacks and capable of adapting to changing attack strategies. Attackers continuously evolve their techniques to evade detection, so it is necessary to incorporate ML and artificial intelligence algorithms that can learn and adapt to new attack patterns because the distributed detection system should be able to differentiate between legitimate traffic surges, such as flash crowds, and actual DDoS attacks. False positives can lead to unnecessary mitigation measures, impacting network performance and user experience.

Another aspect to consider is the coordination and communication between network nodes in the distributed detection system. Efficient exchange of information and collaboration among nodes is essential to achieve accurate and timely detection. Designing a robust and efficient communication framework that minimizes the overhead and latency of information sharing is crucial.

1.4 Dissertation Aim and Objectives

The dissertation aims to develop a distributed detection and prevention system capable of identifying and mitigating DDoS attacks in next-generation networks using ML classifiers.

The objectives of the dissertation are:

A. Implement a feature selection to select the most effective features in three datasets using two feature selection methods.
B. Implement five ML algorithms using the feature selected in step (A) to select the best two algorithms according to accuracy and training time.
C. Develop an ML classifier using a stacked model to combain the two algorithms selected in step (B).
D. Develop a microservice by containerizing the classifier in a docker container and deploying it in the network slice.
E. Develop a dynamic allocation strategy to manage the elasticity of the microservices.
F. Develop a metric server to sense the resource consumption and inform the orchestrator to scale microservices up or down

1.5 Scope of the Dissertation

The proposed work scope can be applied to large organizations that need to protect their network slices (virtual networks) from DDoS attacks by using the combination of ML, virtualization, and dynamic allocation.

1.6 Dissertation Contributions

The contribution of this dissertation can be summarized as follows:

A. Determination of the fastest and most accurate algorithm amongst five ML algorithms using different features to achieve high accuracy when it is evaluated on the CICIDS2019 dataset.
B. Dockrizing the stacked model algorithm to make it able to be deployed in the virtual network by adding a prefilter code to the algorithm code for dropping packets that do not meet the specification of input packets without needing to classify them.
C. Develop a dynamic allocation approach to manage the network resource efficiently.
D. Implement a distributed detection and mitigation technique to improve the security of NGNs.

1.7 The Outlines

This dissertation contains five chapters organized as follows:

Chapter Two: Theoretical Background

This chapter reviews both DDoS attacks and NGNs architecture, the challenges faced by NGNs, and their fundamental characteristics. In addition, the orchestration specification and all the related concepts like virtualization, dockerization, and containerization. Also, the ML and the evaluation of the ML algorithms will be explained.

Chapter Three: Proposed System Design

In this chapter, the proposed system details have been mentioned, including the proposed system and the algorithms that are allocated to detect DDoS attacks using ML techniques.

Chapter Four: Results Discussion and Analysis

The results of the proposed system for DDoS attack detection are discussed. The evaluation results are presented; these results are dependent on accuracy, precision, recall, F-Measure, test time, and tree size.

Chapter Five: Conclusions and Suggested Future Works.

The conclusion was mentioned to DDoS attack detection in the NGNs network and also suggested future works to develop the proposed system.

CHAPTER Two

THEORETICAL BACKGROUND

2.1 Introduction

This section explains the theoretical background of the dissertation including NGNs and its general network architecture with the different types of security measures that are put in place within the platform in order to detect and prevent malicious attacks. It provides an overview of machine learning algorithms used in the dissertation and introduces the concepts of Network slicing, virtualization, orchestration, and dynamic allocation. It also outlines the existing DDoS attacks, detection methods, and software used to establish the attack.

2.2 Next-Generation Networks (NGNs)

It refers to advanced telecommunication networks that have evolved beyond traditional circuit-switched networks to incorporate modern technologies, protocols, and architectures. These networks are designed to meet the growing demands for high-speed connectivity, enhanced quality of service, increased capacity, and support for a wide range of multimedia applications. NGNs came with different features and aspects like:

1. IP-Based Architecture: NGNs are predominantly based on Internet Protocol (IP) and use packet-switching techniques. This enables efficient transmission of data, voice, and multimedia content over a single network infrastructure.
2. Convergence of Services: NGNs bring together various communication services, including voice, data, video, and multimedia, onto a single network platform. This convergence allows for the seamless integration of different communication modes and enhances the user experience.

3. Broadband Connectivity: NGNs provide high-speed broadband connectivity to end-users, enabling faster data transmission and access to multimedia content. This is achieved through technologies like fiber-optic networks, DSL, cable modems, and wireless technologies such as 4G and 5G.
4. Quality of Service (QoS): NGNs prioritize QoS to ensure optimal performance for different types of traffic, including real-time voice and video applications. QoS mechanisms manage network resources effectively, minimizing delays, packet loss, and jitter.
5. Scalability and Flexibility: NGNs are designed to be scalable and adaptable to handle increasing data traffic and evolving technologies. They can accommodate a large number of users and devices while supporting the deployment of new services and applications.
6. Service-Driven Architecture: NGNs adopt a service-oriented architecture (SOA) approach, where services are modular and can be dynamically provisioned, customized, and integrated. This allows for more flexible and efficient service delivery.
7. Security and Privacy: NGNs incorporate advanced security mechanisms to protect against threats and ensure the privacy of users' data. Encryption, authentication, access control, and intrusion detection systems are implemented to maintain network security.
8. Virtualization and Cloud Computing: NGNs leverage virtualization and cloud computing technologies to optimize resource utilization, enhance network management, and enable the deployment of virtualized network functions (VNFs).

9. Internet of Things (IoT) Support: NGNs are designed to accommodate the increasing number of IoT devices and their communication requirements. They provide the connectivity and infrastructure needed for seamless integration and communication between IoT devices.

2.3 Network Slicing

Network slicing is a fundamental concept associated with Next-Generation Networks (NGNs) and represents a significant feature within this framework. The concept revolves around dividing a physical network into numerous distinct and virtualized network segments, each tailored to cater to specific applications, services, or unique customer prerequisites. Each of these network slices functions as an autonomous and self-contained network, custom-designed to precisely address the requirements of the applications or services it serves[48].

The notion of network slicing empowers network operators to assign dedicated resources and establish varying levels of quality of service (QoS), performance metrics, security protocols, and service level agreements (SLAs) for diverse applications or user groups that coexist within the same underlying physical infrastructure. This approach facilitates the efficient utilization of resources, enhances overall network efficiency, and offers the flexibility to support a wide array of use cases with distinct demands and specifications. For instance, an operator could design a network slice optimized to serve autonomous vehicles, prioritizing ultra-low latency and superior reliability. Simultaneously, another network slice might be optimized for extensive Internet of Things (IoT) deployments, emphasizing scalability and energy conservation.

Each individual slice can be managed, configured, and fine-tuned independently, allowing for real-time resource allocation based on evolving demands. Refer to Figure 2.1 for a visual representation of this concept [49].

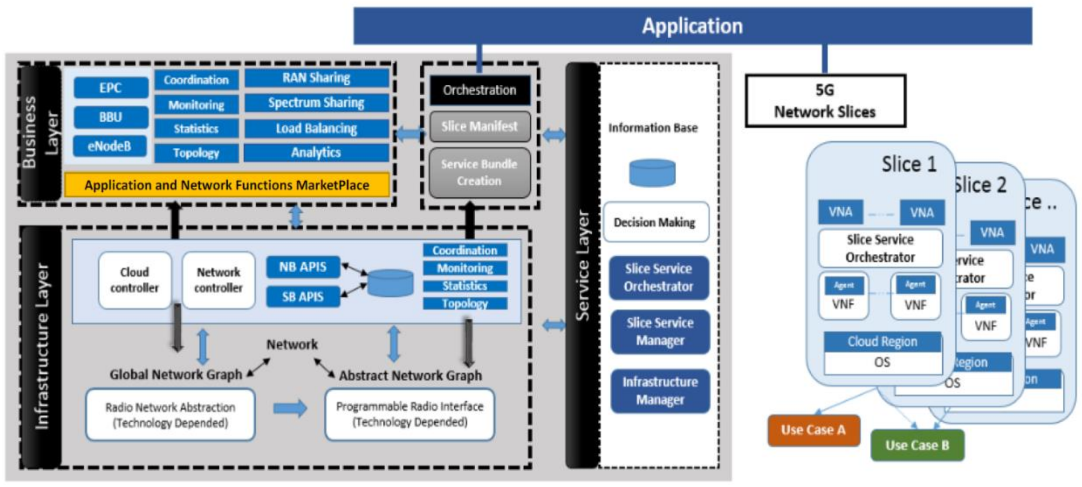


Figure 2-1 NGN Network Slicing Architecture [50]

Next-generation networks provide the underlying infrastructure and architectural enhancements necessary to support network slicing. NGN technologies like SDN and NFV enable the dynamic creation, management, and orchestration of network slices.

2.4 Virtualization

Virtualization refers to the process of creating a virtual (rather than physical) version of a resource, such as a computer system, operating system, storage device, or network. It involves abstracting the underlying hardware or software resources and presenting them in a virtualized form, which can be utilized by multiple users or applications simultaneously. Virtualization technology allows for the efficient utilization and sharing of physical resources, enabling greater flexibility, scalability, and cost savings. It separates the logical functionality or representation of a resource from its physical implementation, providing several benefits:

1. Server Virtualization: In the context of servers, virtualization enables the creation of multiple VMs on a single physical server. Each VM functions as an independent and isolated environment, running its own operating system and applications. Server virtualization allows for efficient utilization of server resources, consolidation of workloads, easier management and deployment of applications, and improved scalability and availability [51].

2. Desktop Virtualization: Desktop virtualization involves running multiple virtual desktop instances on a centralized server or data center. Users can access their virtual desktops remotely from various devices, providing flexibility, mobility, and simplified desktop management. Desktop virtualization enables centralized control, faster provisioning of desktop environments, and enhanced security and data protection [52].

3. Storage Virtualization: Storage virtualization abstracts the physical storage devices (such as hard drives or storage area networks) and presents them as a single virtual storage pool. This allows for easier management, allocation, and optimization of storage resources across multiple systems. Storage virtualization enables features like data migration, thin provisioning, snapshots, and replication for efficient data management and disaster recovery [53].

4. Network Virtualization: Network virtualization abstracts the physical network infrastructure, such as switches, routers, and firewalls, and creates virtual networks on top of it. It allows for the simultaneous operation of multiple logical networks over a shared physical network, providing isolation, security, and flexibility. Network virtualization facilitates the creation of virtual LANs (VLANs), virtual private networks (VPNs), and network overlays, simplifying network management and enabling efficient resource utilization [54].

Virtualization technology has revolutionized the IT industry by improving resource utilization, reducing hardware costs, enabling easier management and scalability, enhancing flexibility, and promoting cloud computing and virtualized infrastructure deployments. It has become a fundamental building block for various technologies like cloud computing, software-defined networking (SDN), and network function virtualization (NFV) [55].

2.5 Virtual Machines

A virtual machine represents a virtualized version of a computer system, with a standard configuration that includes an Operating System and all necessary libraries and binaries. Through the virtualization process, each virtual machine operates independently from others, interacting solely with the host machine and hardware peripherals through the hypervisor. This ensures that any program running within a virtual machine cannot distinguish between its environment and a non-virtualized setting.

Virtual machines can allocate varying amounts of computational resources, making them well-suited for tasks in cloud computing. Cloud service providers can offer users the flexibility to pay based on the resources they utilize. Within a single infrastructure, numerous virtual machines can be accommodated, each with adaptable specifications. Virtual machine instances play a central role in executing cloud computing operations. The standard structure of a virtual machine is illustrated in Figure 2-2. Each instance of a virtual machine contains a duplicate of the guest Operating System, as well as essential libraries and applications. The hypervisor is responsible for initiating and terminating these virtual machines, while also serving as an intermediary between different instances. host OS and the guest systems. Subsequently, the Host OS directly interfaces with the underlying infrastructure [52].

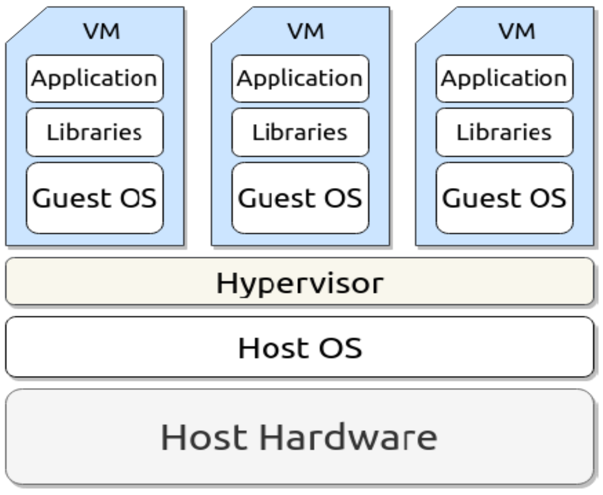


Figure 2-2 Anatomy of a typical virtual machine deployment [56]

2.5.1 Hypervisor

The hypervisor serves as the crucial link connecting virtual machines to their host systems. It can operate directly on the underlying hardware or within a host operating system, depending on the specific situation. In either scenario, the hypervisor is responsible for executing the guest operating systems and interpreting their requests and commands to facilitate system virtualization [52].

2.6 Containers

A container serves as a self-contained virtualized computing environment, resembling a full computer system, similar to virtual machines. Unlike virtual machines, containers operate at the operating system level. This implies that multiple containers on the same host share a common kernel while remaining unaware of each other's presence. Each container is isolated in its own user-space environment with limited hardware access [55].

Containers include essential libraries and dependencies required by their applications, streamlining deployment without concerns about compatibility. Developers can also build new container images based on existing ones, facilitating customization. Using containers provides several advantages akin to virtual machines, such as isolated runtime environments for applications and efficient resource usage. Moreover, containers exhibit a lower overhead compared to equivalent virtual machines [57]. Table 2-3 outlines a comparison between Virtual Machines and Containers.

Containers avoid handling kernel-level tasks, thereby reducing resource consumption. They can also share data through a shared file system when necessary. Containerization enhances application portability across various operating systems and cloud infrastructures. Figure 2-2 depicts a standard containerized deployment, where each container is more lightweight compared to its equivalent virtual machine. The container engine orchestrates the container lifecycle, including startup and shutdown. All containers leverage the Host OS's kernel for kernel-level activities.

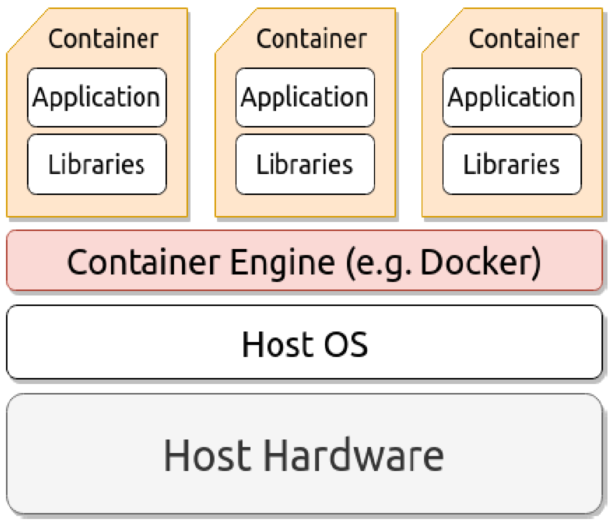


Figure 2-3 Anatomy of a containerized deployment [56]

2.6.1 Container Engine

A container engine functions as software that initiates, halts, and supervises the operation of active container instances. Its role encompasses overseeing containers throughout their complete lifecycle. Frequently, such software incorporates functionalities enabling the generation of a container image based on a supplied specification file or a currently operating instance. Additionally, it possesses the capability to fetch container images from an external repository onto the host machine. Table 2-1 presents a contrast between Virtual Machines (VMs) and Containers [58].

Table 2-1. Comparison between VMs and containers

Features	VM	Container
Performance	Suffer from a low overhead as the instructions from the Guest to the Host OS are translated	Provide close-to-native performance compared to the Host OS.
Startup time	It takes several minutes for VMs to boot up.	Can boot containers in a few milliseconds.
Storage	VMs require even more space as a whole OS kernel as it has to install and run the related programs.	Containers take lower space, as the basic operating system is shared.
Isolation	Hardware isolation.	Operating system isolation
Kernel	Each VM runs its OS	Containers share the same kernel
Benefits	Fully isolated and more secure	Lightweight, Native performance, less memory requirement, more portable
Drawbacks	Heavyweight, limited performance, Large memory requirement, less portable	Higher fault domain and are less secure

2.7 Moving from VMs to Containers in NGNs

NGNs face the challenge of effectively utilizing infrastructure as a service to ensure flexibility, security, reliability, and ultimately, profitability. What began as an experiment involving the utilization of NFV to operate VMs on hardware has rapidly evolved into VMs hosted on shared computing and storage farms. The introduction of an orchestration layer was crucial to minimize operational complexities. However, achieving optimal resource utilization with VMs has proven to be a persistent challenge, necessitating a transition to a Container architecture (CNFs). The architecture of the 5G mobile network control plane has the potential to adopt a hybrid structure that combines cloud-native applications and virtualization [59]. Illustrated in Figure 2-4, the network virtualization approach transforms conventional network appliances, previously reliant on non-standard hardware, into software-based virtual machines deployed on standardized equipment. Traditional monolithic network functionalities have been modularized into smaller microservices and delivered as containers using the cloud-native methodology, both within public and private clouds [60]. These microservices containers are effectively orchestrated and automatically deployed through Continuous Integration and Deployment (CI/CD) processes. Independent software providers are currently responsible for delivering these smaller microservices, a departure from their previous role in providing comprehensive network operations.

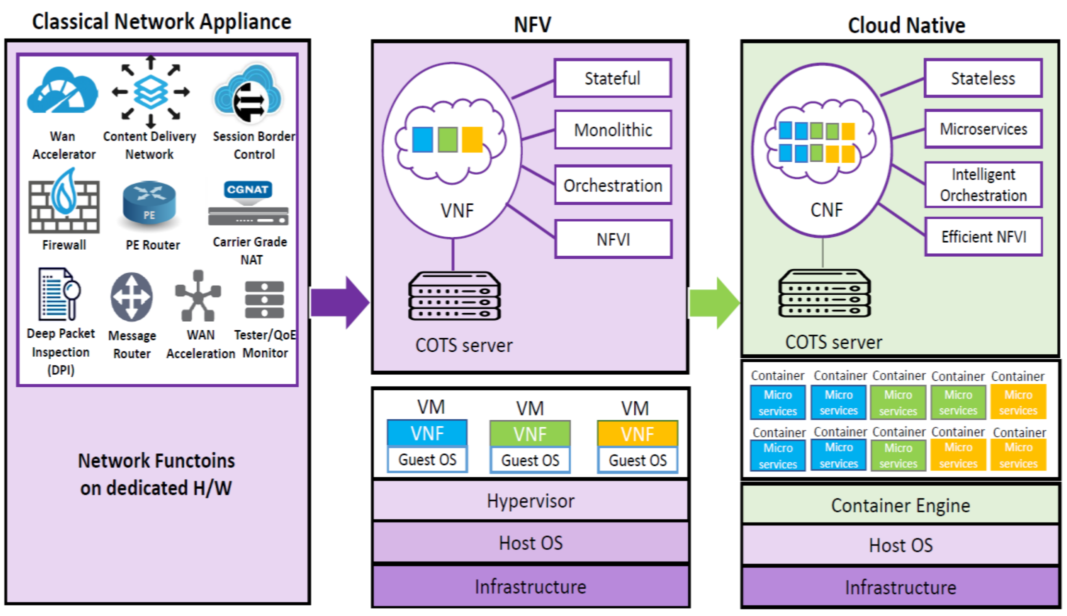


Figure 2-4 Monolithic architecture, VNF virtualization architecture, and CNF Cloud architecture [58]

2.8 Monolith

To comprehend the reasons underlying the growing popularity of microservices, it's crucial to first grasp the concept of a monolith. In the realm of server-oriented programming languages, there exists the capability to segment various elements of a program into discrete modules. This modular approach serves to simplify the intricacy of both code and application by managing distinct functions of the application. A monolith, on the other hand, denotes a software application that lacks the ability to operate these modules autonomously. In simpler terms, each module relies on other modules within the server to fulfill their designated tasks. Visualized in Figure 2-5, a monolithic architecture depicts a single machine responsible for handling each overlapping aspect of the application [60].

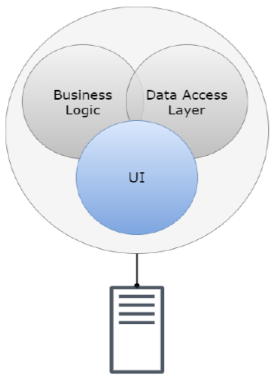


Figure 2-5 Illustration of a Monolithic Architecture

2.8.1 Issues

The evolution of microservices aims to resolve the challenges associated with the monolithic server architecture. Initially, the complexity of applications using a monolithic approach becomes a concern. Due to interdependencies among modules, the architecture becomes intricate, making maintenance and future development challenging [60]. Making changes in one module often necessitates modifications in others, leading to a cascading effect. This complexity also complicates bug tracking, as identifying the origin of a bug might involve traversing a convoluted path across modules. Moreover, the architecture's intricacy introduces scalability issues. Scaling up to handle increased tasks is straightforward by creating new instances of the application. However, this becomes cumbersome and costly for extensive systems, even if only specific modules are affected by the increased workload.

From a developer's perspective, the monolithic approach presents several suboptimalities. For instance, when developing a service requiring extensive testing, progress can be sluggish due to the need to reboot the entire application upon changes in any module. This can be especially problematic for larger systems, demanding substantial time for rebooting. The uniform programming language requirement across the application further impedes developers. While different modules might perform optimally in distinct programming languages, the architecture forces a single language selection, potentially leading to inefficiencies or undue complexity. The same constraints extend to libraries and dependencies, where discrepancies arising from updates for one module can affect others utilizing older configurations.

2.9 Microservices

Microservices attempt to tackle these challenges by establishing modularity and independence among individual components, which are then distributed across a network Figure 2-6. In this arrangement, each distinct component functions as a microservice, offering a range of advantageous capabilities. Additionally, microservices have the potential to significantly lower infrastructure expenses, as evidenced by Villamizar et al.'s cost analysis [58], [60]. Their findings indicate that adopting a Microservices Architecture (MSA) instead of a monolithic approach could lead to a substantial 17% reduction in infrastructure costs for software application providers.

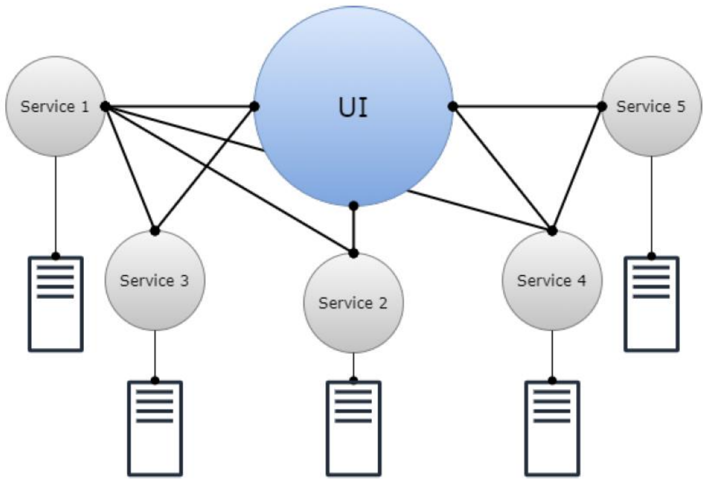


Figure 2-6 Illustration of an MSA module

2.9.1 Architecture

Microservices Architecture (MSA) allows for the distribution of various services across separate machines, each operating as distinct, standalone services. For instance, consider an e-commerce application:

different microservices could handle buying clothes, processing payments, retrieving clothing images, and managing logins and authentication. This modular approach ensures that each microservice focuses solely on a specific task. The isolation means that an image-related microservice doesn't require knowledge of payment or authentication processes, and vice versa. Intercommunication among microservices is achieved through message passing.

For instance, imagine the payment microservice needing to validate a user before processing a payment. It interacts with the authentication microservice for this purpose. A visual representation of MSA is provided in Figure 2-6. This architecture addresses concerns raised by the monolithic approach. Firstly, system components are simpler, with each microservice handling a limited set of related functions. This simplification eases application development and maintenance since each microservice can be worked on independently, maintaining a high degree of simplicity. Moreover, updates require restarting only the affected microservice, not the entire system as in a monolithic setup.

Moreover, microservices execute independently, enabling developers to use different programming languages and frameworks for each microservice. This allows specialization for efficiency and simplicity. The only stipulation is that microservices adhere to the established message-passing protocol. As long as a microservice can send and receive messages as intended, it can employ any programming language and framework [61].

2.10 Orchestration

Traditional grid systems often rely on batch engines like Condor [62]for scheduling tasks and software on distributed computing clusters. However, running containers instead of regular batch jobs demands contemporary orchestration tools that enable users to manage virtualized workloads. A few widely used options include Google Kubernetes, Apache Mesos, and Docker Swarm. These open-source solutions offer features for deploying, scaling, and overseeing applications that are containerized.

A. Google's Kubernetes [63] is a creation of Google and is said to be the outcome of extensive experience in managing production workloads. Designed to handle billions of containers weekly, Kubernetes offers features like horizontal scaling, automated updates and rollbacks, storage coordination, automatic recovery, service identification, load distribution, secret and configuration control, and batch processing.

B. Apache Mesos [63] was constructed with a design resembling that of the Linux kernel. It establishes the Mesos kernel on each machine within a cluster, granting applications like Hadoop, Spark, Kafka, and Elasticsearch access to an application programming interface (API). This API facilitates resource management and job scheduling in data centers. Mesos utilizes Linux cgroups to ensure separation for CPU, memory, I/O, and file system isolation. As a result, it emerges as an alternative container engine to Docker, but it features a distributed orchestration system for containers.

C. Docker Swarm [64], a feature of the Docker engine designed for distributed deployments, offers integrated cluster management through the command line interface (CLI). It functions as a native clustering system within Docker. For the Arhuaco architecture, Docker Swarm was chosen as the initial supported container management system. This choice expands upon the LC concept by enabling the complete distribution of the entire operating system to run atop a host OS. As an example, it permits the execution of a containerized version of Ubuntu on a Debian base. Docker Swarm, being integrated with Docker, the selected container engine for this study, was a logical decision. This integration facilitates rapid prototyping and experimentation within the testing environment.

Upon investigating the isolation elements within the suggested framework, an explanation is provided regarding how containers offer monitoring data that can be leveraged to ascertain the conduct of an application.

2.11 Auto Scaling

Hardware virtualization includes a capability for adjusting the resources of a container while it's active. This characteristic enables the allocation of extra CPUs, primary memory, storage, and network bandwidth to a container as needed. Furthermore, it allows for the removal of allocated resources when they're inactive or unnecessary. Numerous providers employ this resource allocation method through auto-scaling web services, enabling cloud users to determine resource requirements based on usage or similar metrics. This same capability is also extended to adding more instances of containers across additional physical servers and halting them when not required. Machine-level scaling (vertical scaling) and data center or cloud-level scaling (horizontal scaling) represent two pivotal aspects of utility computing. Scalability is achieved by distributing an application across numerous physical servers in the cloud [65]. Fast interconnects and ample storage drive this scalability. Operating system virtualization plays a significant role in container scalability. The rapid cloning and deployment of containers contribute to swift scalability. When the need arises, cloned containers can be launched on alternative servers to share the workload. Scalability is further enhanced by the live migration of containers, which permits a running virtual server to seamlessly transition to a larger physical server with minimal downtime, ensuring uninterrupted and scalable operations.

2.12 Machine Learning

This dissertation investigates the application of ML for detecting unauthorized access within a Kubernetes Cluster. ML, a branch of AI, seeks to enable machines to make decisions autonomously, devoid of explicitly programmed instructions. This entails the development of adaptable algorithms that can produce precise outcomes. In the context of this study, algorithms can be formulated to recognize malicious behavior within a Cluster. Multiple approaches exist for implementing a learning algorithm [66].

2.12.1 Supervised Learning

Supervised learning is a type of machine learning where an algorithm learns from labeled training data to make predictions or decisions. In supervised learning, the algorithm is provided with input data and corresponding output labels, and its task is to learn the mapping between the inputs and outputs. The goal is to enable the algorithm to make accurate predictions or classifications on new, unseen data. Supervised learning can be further categorized into two main types:

1. Classification: In classification, the algorithm's task is to assign a category or label to each input data point. For example, classifying emails as spam or not spam, or identifying whether an image contains a cat or a dog.
2. Regression: In regression, the algorithm's task is to predict a continuous numerical value based on the input data. For example, predicting house prices based on features like square footage, number of bedrooms, etc.

Supervised learning is widely used in various applications, including image recognition, natural language processing, fraud detection, medical diagnosis, and many others. It relies on having a labeled dataset to train the algorithm effectively, and its success depends on the quality and representativeness of the training data

2.12.2 Unsupervised Learning

Unsupervised learning is a machine learning paradigm in which an algorithm is tasked with finding patterns or structures within a dataset without being explicitly provided with labeled target outcomes. Unlike supervised learning, where the algorithm is trained on labeled data to make predictions or classifications, unsupervised learning operates on unlabeled data, seeking to uncover inherent relationships, groupings, or representations within the data. In unsupervised learning, the algorithm's goal is often to discover the underlying structure of the data or to reduce its dimensionality while revealing meaningful patterns. There are two main types of unsupervised learning techniques:

1. Clustering: Clustering algorithms group similar data points together based on their intrinsic characteristics. The goal is to partition the data into clusters or groups where data points within the same cluster are more similar to each other than to those in other clusters. Common clustering algorithms include K-Means, Hierarchical Clustering, and DBSCAN.
2. Dimensionality Reduction: Dimensionality reduction techniques aim to reduce the number of features or variables in the data while preserving important information and patterns. This can help simplify data analysis and visualization. Principal Component Analysis (PCA) and t-SNE (t-Distributed Stochastic Neighbor Embedding) are examples of dimensionality reduction methods.

2.12.3 Reinforcement Learning

In the context of the title, the algorithm gains knowledge through a system of rewards and penalties. It is rewarded for correct actions and penalized for incorrect ones. This mechanism enables the algorithm to autonomously learn and base its decisions on previous experiences, much like how a young creature learns from its surroundings and adjusts its behavior accordingly.

2.12.4 Machine learning algorithms

Six machine learning algorithms are implemented in this dissertation, five of them are used without any modification and the sixth one represents a combination of two of them. The algorithms are identified one by one in the following section:

A. **J48 or C4.5**: J48 [67], commonly referred to as C4.5, is a widely used machine learning technique integrated into the decision tree algorithm. Employing the concept of entropy, this approach builds a decision tree based on a provided training dataset. It differs from IDE3 in that it creates a decision tree, and unlike IDE3, J48 or C4.5 accommodates both continuous and categorical attributes.
B. **REP Tree:** The REPT algorithm [68], derived from the C4.5 algorithm, is a rapid decision tree method suitable for constructing both classification (categorical result) and regression (continuous outcome) trees. It constructs a decision or regression tree by utilizing information gain or variance, and then refines its structure using reduced-error pruning, including back-fitting.
C. **Random Forest (RF):** Due to its utilization of an ensemble of Decision Trees, the RF [69] classifier attains remarkable predictive accuracy for classification tasks. The multitude of decision trees collaborates in assigning categories, with each tree individually determining the appropriate class for a new instance. The final class assignment is based on a majority vote among the trees. As the quantity of trees contributing to the decision-making process increases, precision is enhanced. Before employing the classifier on datasets, it is essential to specify the desired number of trees.
D. **Partial Decision Tree (PART):** Numerous methods exist for generating rules from decision trees. C4.5 and RIPPER [70] represent the primary approaches for rule learning, both involving

a two-stage process. Initially, C4.5 generates an initial rule set, followed by refining the set through an intricate optimization phase that eliminates individual rules. Similarly, RIPPER refines rules by adjusting individual components. The combination of these approaches, known as Partial Decision Tree (PART) [71], forms a unified scheme that doesn't necessitate complex optimization stages. Merging C4.5 and RIPPER in PART is uncomplicated, effective, and direct. This entails constructing a pruned decision tree for the present instance set. The most extensive coverage leaf is transformed into a rule, and the tree is discarded after removing covered instances from the training dataset. Employing a separate-and-conquers strategy, this process is repeated for all training dataset instances. Notably, the PART algorithm produces rule sets that surpass the accuracy of RIPPER's, while matching C4.5's accuracy level. Furthermore, PART's rule sets are more concise compared to C4.5's rule sets.

E. **Multilayer perceptron (MLP)** is a type of artificial neural network that is composed of multiple layers of neurons connected. It is used for supervised learning tasks such as classification and regression. MLP [72] networks are composed of an input layer, one or more hidden layers, and an output layer. Each layer is made up of neurons that have weighted connections to the neurons in the layers before and after it. The neurons in the hidden layers use an activation function to process the inputs and generate them.

F. **Stacked algorithm:** A stacked algorithm [66], often referred to as a stacked ensemble or stacked generalization, is a machine learning technique that combines the predictions of multiple individual models (base models) to improve overall predictive performance. The idea behind stacking is to leverage the strengths of different models and reduce their weaknesses by combining their outputs. The idea behind stacking is to capture the diverse perspectives of individual models and allow them to compensate for each other's weaknesses. This often leads to improved performance and generalization compared to using a single model. It's important to note that while stacking can be very effective, it can also be computationally expensive and requires careful tuning of various components, such as selecting the right base models, deciding on the architecture of the meta-model, and handling potential issues like overfitting. Additionally, stacking requires a larger amount of data compared to simpler algorithms due to the need for the holdout set and meta-model training data.

2.13 Feature Selection Methods Used in this Study

One of the most common problems researchers encounter is choosing which features are most important and thus relevant for use in detecting attacks. Feature selection is critical because it affects how well the system works. Too few features may lead to subpar detection accuracy, while too many may lead to excellent detection accuracy at the expense of an overly complex system that eats up more resources. This study employed two attractive feature selection techniques

1. **Wrapper Method:** Wrappers [72] assess subsets of features based on their performance using a modeling algorithm, which is treated as an external evaluator without revealing its internal

workings. In the case of classification tasks, wrappers evaluate subsets according to the classifier's performance, such as Naïve Bayes or SVM. Similarly, for clustering tasks, the assessment relies on a clustering algorithm's performance, like K-means. This evaluation process is iterated for each subset, with the subset generation strategy mirroring that of filters.

Wrappers differ from filters in that they tend to be slower at identifying satisfactory subsets due to their reliance on the resource requirements of the modeling algorithm. Consequently, the selected feature subsets often display a bias towards the specific modeling algorithm used for evaluation, even when employing cross-validation. To ensure a dependable estimation of the generalization error, it is advisable to incorporate an independent validation sample and an alternative modeling algorithm after identifying the final subset.

However, empirical evidence has demonstrated that wrappers yield subsets with superior performance compared to filters. This improvement can be attributed to the fact that wrappers employ actual modeling algorithms for evaluation. In practice, wrappers can adopt a variety of search strategies and modeling algorithms, but they are particularly well-suited for greedy search strategies and swift modeling algorithms, such as Naïve Bayes, linear SVM, and Extreme Learning Machines.

2. **Filter Method:** The filter feature selection method [73] is one approach to accomplish this. It involves evaluating the

individual features independently of the machine learning algorithm being used and selecting the most relevant ones based on some predefined criteria. The filter feature selection method generally consists of the following steps:

a. Feature Scoring: Each feature is scored or ranked using some statistical measure or heuristic. The goal is to quantify the relevance of each feature to the target variable. Common scoring methods include:
    - Chi-squared (χ^2): Measures the dependence between categorical variables.
    - Information Gain or Mutual Information: Measures the amount of information gained about the target variable from knowing the feature.
    - Correlation: Measures the linear relationship between numerical variables and the target.
    - ANOVA F-statistic: Measures the variability between different classes in the target variable.

b. Ranking: After scoring, features are ranked based on their scores in descending order. The higher the score, the more relevant the feature is considered.

c. Feature Selection: A threshold or a fixed number of top-ranked features is chosen based on domain knowledge or experimentation. Features that meet the threshold are selected as the final subset of features.

d. Training Model: A machine learning model is trained using only the selected subset of features.

2.14 Distributed Denial of Service Attacks

An online service or network is at risk of a DDoS attack in terms of security. This dissertation addresses the study issue of DDoS attacks and offers several frameworks for its resolution. DDoS assaults try to reduce a service's availability by using up all of the network or computational resources available for traffic or computation/processing, which prevents authorized users from using the services of their victims [74].

2.14.1 Launching of DDoS Attack

Initiating a Distributed Denial of Service (DDoS) assault can be initiated through various means. However, the predominant approach involves an attacker flooding a targeted server with a continuous stream of packets, depleting vital resources and obstructing legitimate users' access to these resources. Another prevalent technique entails transmitting a handful of malicious packets capable of inducing server paralysis or reboot. Additionally, commandeering machines within the victim's network to exhaust critical resources is another method to disrupt a service's functionality. This results in the network becoming inaccessible for both internal and external services. Numerous alternative strategies exist for executing such attacks, rendering their anticipation challenging. Frequently, these methods are only discerned post-factum, once the attacks have already been executed [74].

2.14.2 DDoS Attack Types

Distributed Denial of Service (DDoS) attacks come in various types, each with its distinct methods and characteristics. DDoS attacks aim to overwhelm a target's network, system, or application resources, rendering them unavailable to legitimate users. The common types of DDoS attacks [75] are:

1. Volumetric Attacks:
   a. UDP Flood: Attackers send a large volume of User Datagram Protocol (UDP) packets to flood the target's network, consuming its bandwidth and causing network congestion.
   b. ICMP Flood: Attackers flood the target with ICMP (ping) requests, consuming network resources and overwhelming the target's ability to respond.
   c. TCP SYN Flood: Attackers send a high volume of TCP SYN requests, exhausting the target's resources as it tries to establish connections that are never completed.
2. Protocol Attacks:
   a. DNS Amplification: Attackers exploit insecurely configured DNS servers to send large amounts of DNS response traffic to the target, amplifying the attack's impact.
   b. NTP Amplification: Attackers exploit Network Time Protocol (NTP) servers to generate a high volume of traffic directed at the target.

   c. SSDP/UPnP Reflection: Attackers use misconfigured SSDP (Simple Service Discovery Protocol) or UPnP (Universal Plug and Play) devices to reflect and amplify attack traffic.

3. Application Layer Attacks:
    a. HTTP/HTTPS Flood: Attackers flood the target's web server with a high volume of HTTP/HTTPS requests, overwhelming its resources.
    b. Slowloris: Attackers send partial HTTP requests to the target's web server, keeping the connections open and consuming server resources, leading to slow response times or service unavailability.
    c. Application Layer (Layer 7) Attack: Attackers focus on targeting specific application vulnerabilities to disrupt or exhaust application resources.
4. State-Exhaustion Attacks:
    a. TCP State-Exhaustion Attack: Attackers target stateful network devices, such as firewalls or load balancers, to exhaust their connection tracking tables or other resources.
    b. Out-of-State TCP Flood: Attackers send non-standard TCP packets to exhaust stateful device resources.
5. Zero-Day Attacks:
    a. Attackers exploit vulnerabilities in software or hardware for which no patches or fixes are available, making these attacks harder to defend against.
6. Amplification Attacks:
    a. Attackers leverage servers or devices that respond with larger volumes of traffic than the original request, amplifying the attack's impact. Examples include DNS, NTP, and SSDP amplification attacks.

7. IoT Botnet Attacks:
    a. Attackers compromise and control large numbers of Internet of Things (IoT) devices, forming botnets to launch coordinated DDoS attacks.

2.15 Intrusion Detection

Intrusion detection mechanisms offer several benefits to system administrators:

- Alerts are triggered upon detecting malicious behavior. This empowers administrators to swiftly secure critical systems through backups or offline modes to thwart data loss or damage.
- The deployment of honeypots [76] can divert attackers away from genuine targets, effectively consuming their time and resources.
- Administrators gain the capability to monitor intruders' activities within the system and record their actions. This yields vital insights into compromised system areas and the effectiveness of security measures.

2.15.1 Honeypots

A honeypot represents a defensive measure against system attacks. It refers to a deliberately created system that is intentionally targeted by malicious actors [76].

Honeypots come in different forms based on the intended attack mitigation and the objectives of the deployer. To capture and monitor intruders, a complex design is employed, closely simulating a genuine system and encouraging extensive engagement from attackers.

Alternatively, for capturing login attempts for logging purposes, a simpler design suffices.

Beyond defense, honeypots serve as a valuable research tool. Security researchers utilize them to analyze ongoing attacks, record typical attacker behaviors within compromised systems, document an attacker's keystrokes, and identify emerging security threats.

2.15.1.1 Advantages

Enticing an intruder to focus on a honeypot presents several benefits. To start with, the honeypot can divert the attacker's attention away from a potentially more susceptible system that requires safeguarding. This diversionary tactic allows for the vulnerable system's temporary shutdown or backup, preventing compromise. Additionally, the honeypot can be purposely configured to impede the attacker's progress, possibly through the implementation of an extensive data volume for exploitation. Lastly, the attacker's techniques can be recorded and employed as a valuable resource for enhancing the security of future system designs.

2.15.1.2 Challenges

An essential hurdle in designing honeypots is ensuring their appearance as genuine systems to attackers. This challenge is further complicated by the divergence between what seems authentic to a human attacker manually initiating an assault and what meets the criteria for legitimacy according to malware. Human attackers might prioritize human-readable file names and indicators of authentic system use, such as newly generated files containing actual data. In contrast, automated attackers may not emphasize filenames or proof of genuine usage as much as humans do, but they could swiftly compare the target environment with known honeypot setups to ascertain its honeypot nature.

In [77], researchers investigated the creation of a botnet with the ability to autonomously identify whether a compromised system was a honeypot. They accomplished this by trying to send malicious packets from an infected target to a controlled destination. In cases of failure or if attack commands were ignored, the botnet effectively removed itself from the compromised system without drawing attention.

The authors delved further into this concept by introducing a two-stage infection process. In this approach, the target is initially infected by a "spearhead." The main attack unfolds and the actual payload is deployed only if the spearhead can validate the absence of honeypot characteristics in the target. This approach capitalizes on the fact that many individuals or organizations deploying honeypots would prefer not to be associated with real-world attacks, thus safeguarding their resources.

2.15.2 Honeynets

Honeynets are an advancement of the honeypot concept, offering enhanced monitoring capabilities for active attacks on a system while presenting a more alluring target to attackers. A honeynet typically employs an arrangement of high-interaction honeypots, configured in a realistic network setup. This design encompasses multiple tiers of data gathering. By deploying honeypots that replicate intricate network environments, including mock SQL databases, fictitious web servers, and simulated WNs, a broader spectrum of attack types can be enticed into targeting the honeynet.

in [78], the author outlines the foundational deployment of a honeynet, referred to as "Gen I." This configuration entails a straightforward and self-contained ecosystem, primarily suited for research purposes. Its primary aim is to lure attackers into engaging with the honeynet to scrutinize the mechanics of their attacks. The setup primarily consists of a network primarily comprised of honeypots, supplemented by tools like the Snort Intrusion Detection System and a logging server. Consequently, its practicality for safeguarding a production network is limited. The architectural representation is depicted in Figure 2-7, with the honeypots indicated in yellow.

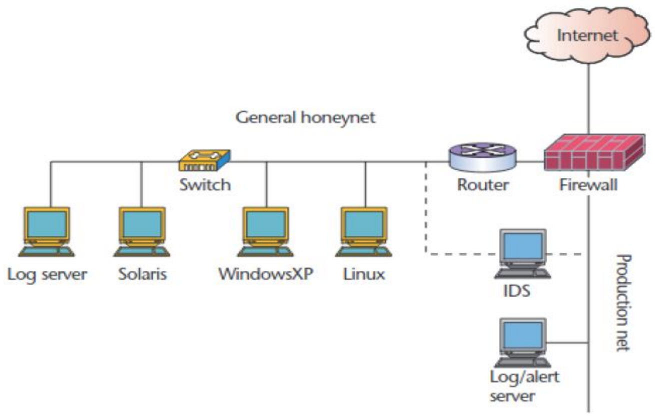


Figure 2-7 Gen I Honeynet Design

The author proceeds to outline a more advanced configuration for a honeynet, referred to as a "Gen II" framework. This upgraded design facilitates the containment of malicious network activity within an internal honeynet. An additional noteworthy aspect of this approach involves the incorporation of a layer 2 bridge named the "Honeynet sensor." This bridge permits incoming packets from attackers to move unrestrictedly but constrains outgoing traffic to impede the attacker's ability to disseminate further attacks. The Honeynet sensor also offers the capability to capture packets and log the keystrokes of the attacker.

This architectural concept is illustrated in Figure 2-8, where the honeypots are represented by yellow machines, and the production machines are depicted in grey. The Honeynet sensor operates as a bridge that adeptly sifts through malevolent traffic, thereby retarding the progress of outbound attacks.

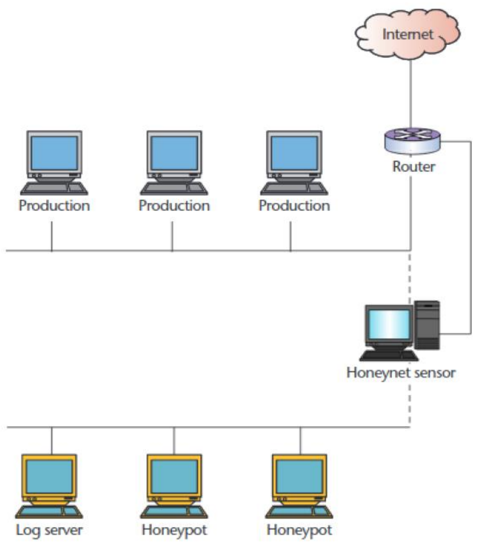


Figure 2-8 Gen II Honeynet Design

2.15.3 Signature-Based Intrusion Detection

In signature-based intrusion detection [79], network traffic and system behavior are matched against a database of established malicious or questionable patterns. This strategy resembles the method employed by commercial antivirus software.

However, a drawback of this approach is its reliance on recognizing known attacks, leaving it unable to identify novel attack variants. Additionally, a resourceful attacker can mask their actions and evade detection within a purely signature-based framework.

2.15.4 Anomaly-Based Intrusion Detection

Unlike signature-based techniques, anomaly-based intrusion detection [79] doesn't require prior knowledge of a specific attack to identify it. Instead, it utilizes heuristics to assess whether the current network traffic deviates suspiciously from the expected pattern.

Anomaly-based methods might entail a training phase during which the system familiarizes itself with the usual traffic behavior within the system. This could involve the utilization of a machine learning model to recognize typical patterns.

2.16 Performance Evaluation

The performance and effectiveness of the proposed system are evaluated by several metrics. A confusion matrix is a table that is often used to describe the performance of the proposed system. A binary confusion matrix is described in Table (2-2).

Table 2-2 A Binary Confusion Matrix

		Predicted Classes	
		Anomaly	Legitimate
Actual Classes	Anomaly	True Positive (TP)	False Negative (FN)
	Legitimate	False Positive (FP)	True Negative (TN)

- TP: The count of anomaly records is accurately categorized.
- TN: The count of normal records is accurately categorized.
- FP: The count of normal records is inaccurately categorized.
- FN: The count of anomaly records is inaccurately categorized.

For the evaluation purpose, Accuracy, Precision, Recall, and F1-score are applied. These metrics are calculated as follows:

Accuracy is showing the percentage of true detection over total traffic trace. To calculate accuracy, through the proportion between the number of packets correctly classified whether these packets are normal or attack over a total number of the packets classified by the proposed system correctly and incorrectly, as shown in Equation (2-1).

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad \dots\dots\dots\dots\dots\dots\dots\dots\dots.\dots (2.1)$$

$$\text{Recall} = \frac{TP}{TP+FN} \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (2.2)$$

$$\text{Precision} = \frac{TP}{TP+FP} \tag{2.3}$$

$$\text{F Score} = 2\,\frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.4}$$

The efficiency of the proposed system has been evaluated through several measures based on the confusion matrix. These measures have been mentioned in [80].

2.16.1 Attack Detection Rate (ADR):

To calculate the Attack Detection Rate (ADR), the ratio between the total number of attack detected packets through the proposed system to the total number of attack packets, as shown in Equation (2.5).

$$\text{ADR} = \frac{\text{attack detected packets}}{\text{total number of attack packets}}$$
$$= \frac{\text{T}N}{\text{TN} + \text{FN}} \ \dots\dots\dots\dots\ \dots (2.5)$$

2.16.2 False Alarm Rate (FAR):

To calculate the False Alarm Rate (FAR), the proportion between the number of normal packets incorrectly categorized as attack packets to the total number of actual normal packets is also named the false discovery rate (see Equation 2.6).

FAR = (number of misclassified normal packets)/(total number of normal packets)

$$\text{FAR} = \frac{FP}{FP+TP} \tag{2.6}$$

2.16.3 False Positive Rate (FPR):

To calculate the False Positive Rate (FPR), through the proportion between the number of normal packets incorrectly categorized as attack packets to total packets classified as attack packets, as shown in Equation (2.7).

$$\text{FPR} = \frac{FP}{FP+TN} \tag{2.7}$$

Finally, to succeed in overloading detection algorithms, must be high accuracy, low FAR, and high ADR.

2.17 Tools and Software Used in the Proposed Work

Several tools and software are used in this dissertation to achieve the final goal. The tools and software are:

2.17.1 Docker Container

A Docker container [55] is a compact, isolated, and transportable software bundle that encapsulates an application along with all its prerequisites, such as libraries, frameworks, and runtime settings. Containerization technology establishes a uniform and reproducible environment for executing applications. Docker containers find their foundation in Docker, an open-source platform enabling developers to automate application deployment and administration within containers.

Containers bring about isolation at the process level, allowing applications to operate autonomously without causing interference. Their exceptional portability allows deployment on any Docker-compatible system since they encompass all necessary elements. In contrast to conventional virtual machines, Docker containers are lightweight and impose fewer resource demands. They leverage the host system's kernel, leading to swift startup periods and effective resource utilization. Multiple containers can coexist on a single host with minimal performance overhead.

Moreover, Docker containers adopt a declarative approach to specify an application's surroundings and prerequisites through a Docker file. This file outlines the foundation image, software components, settings, and stages to construct the container. This methodology empowers developers to produce consistent builds, ensuring application uniformity across diverse settings.

2.17.2 Kubernetes

Kubernetes [81], an open-source platform for orchestrating containers, streamlines the deployment, scaling, and administration of containerized applications. It offers a

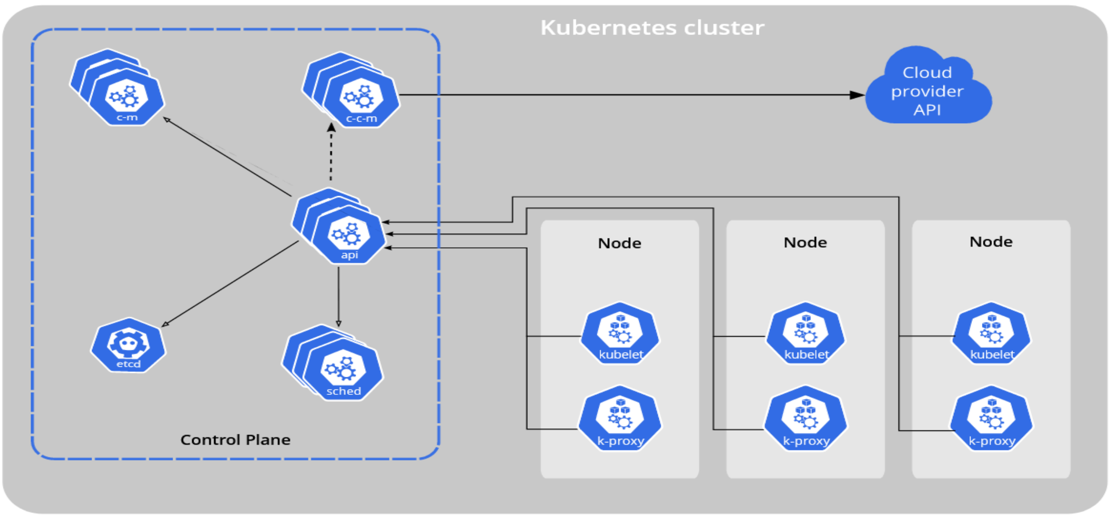


Figure 2-9 the components of a Kubernetes cluster

framework for effectively running and coordinating numerous containers across a server cluster, simplifying the management of intricate distributed systems.

In a Kubernetes cluster, the architecture follows a master-worker model. The master node (MN) functions as the control center, overseeing and orchestrating the allocation of cluster resources and the scheduling of containers. Meanwhile, the worker node (WN), often referred to as minions, is responsible for hosting the actual containers and executing workloads.

The platform boasts an extensive array of features, encompassing service discovery, load balancing, automatic scaling, seamless updates and rollbacks, storage management, self-healing mechanisms, and more. These functionalities collectively empower applications to achieve high availability, scalability, and robustness.

Kubernetes administers applications through declarative configuration files labeled as "Kubernetes objects." These objects succinctly define the desired application state, covering aspects like containers, networking, storage, and additional resources. Additionally, Kubernetes exposes a RESTful API, which grants users the ability to interact with the cluster, govern resources, and manage the intended system state. This API serves as a programmatic interface for automating and overseeing various Kubernetes operations.

2.17.2.1 Kubernetes Architecture

The core framework of Kubernetes consists of a Cluster, illustrated in Figure 2-10. This Cluster is composed of an MN and multiple WNs.

Each Node houses an operational kubelet process, facilitating smooth communication among the Nodes present within the Cluster.

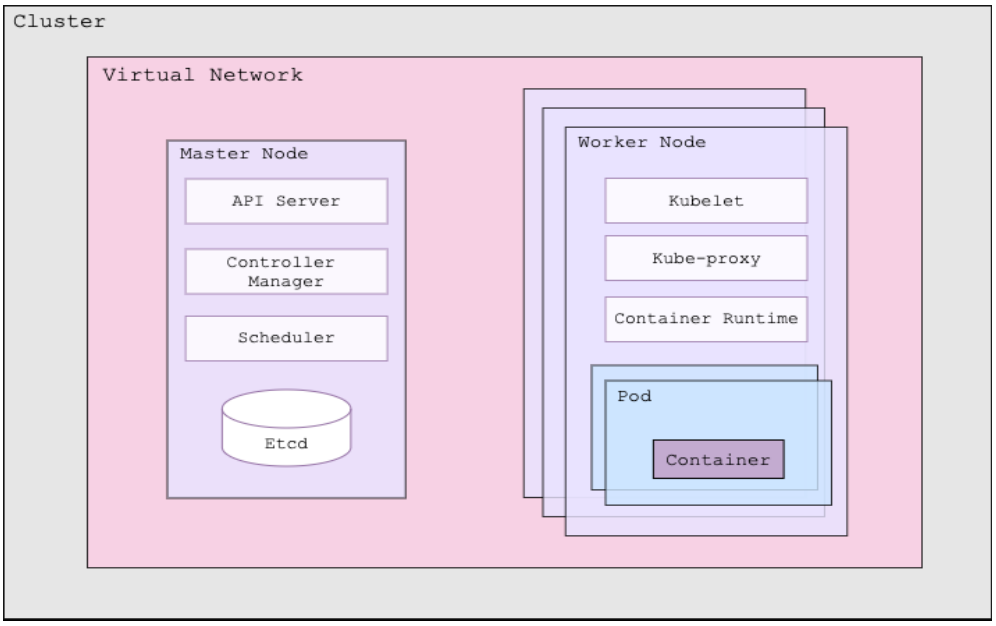


Figure 2-10 illustrates a broad perspective of a Kubernetes Cluster, which comprises a central MN and numerous WNs.

2.17.2.2 Master Node

In Kubernetes, the MN (also known as the control plane) is a critical component responsible for overseeing and managing the entire Kubernetes cluster. It contains essential services and components that control the cluster's behavior, orchestrate container deployments, and ensure proper communication between different parts of the system. The MN plays a central role in maintaining the desired state of the cluster and managing its operations. The API server is the front-end for the Kubernetes control plane. It exposes the Kubernetes API, which allows users, administrators, and other components to interact with the cluster. It serves as the entry point for all communication with the cluster. Where

The scheduler is responsible for assigning work (in the form of pods) to WNs in the cluster. It makes decisions about which node a pod should run on based on factors like resource availability, workload, and affinity/anti-affinity rules.

The controller manager in MN is a collection of controllers that monitor the state of the cluster and take actions to maintain the desired state. It includes controllers such as the Node Controller (ensuring nodes are healthy), Replication Controller (maintaining the desired number of pod replicas), and Service Account & Token Controller (managing security credentials).

Finally, the Etcd is a distributed key-value store that holds the entire configuration and state information of the cluster. It serves as the single source of truth for the cluster's data, including details about nodes, pods, services, and more.

2.17.2.3 Worker Node

WNs, referred to as the Data Plane, house multiple Pods containing containerized applications. A Cluster can comprise several WNs, managed by the MN. Much like the MN, the WN has essential components for its operation.

The Kubelet, an agent present on all Nodes including the Master, guarantees the well-being of containers within Pods, focusing primarily on WNs due to the recommendation against running Pods within the MN. It serves as an intermediary between the Master and WNs. Monitoring the API Server on the Master, the Kubelet carries out commands associated with its Pods.

Kube-proxy, a network proxy on all Nodes, upholds network configurations of Node-bound Pods. It ensures communication between Node-bound Pods and other Pods or Nodes using techniques like iptables, proxy servers, and traffic forwarding. Additionally, Kube-proxy facilitates Kubernetes' Services Concept, allowing application exposure as Network Services. This involves tagging Pods to maintain consistent identification despite IP address changes.

Container Runtime manages container execution within Nodes. Kubernetes supports multiple Runtimes, including Docker, Containers, CRI-O, and other Kubernetes Runtime Interfaces.

A Pod, the smallest manageable unit in Kubernetes, serves as an abstraction layer for single or multiple containers needing cooperation. Containers within a Pod share settings, storage, network settings, IP addresses, and port space. Inter-Pod communication occurs through local hosts, while external communication necessitates shared network ports. Node-assigned Pods have been under Node management since creation, ending when the assigned Node fails. Ephemeral by nature, Pods frequently restart and are recreated with the same name but a unique ID and IP address if necessary.

2.17.2.4 Networking in Kubernetes

A Kubernetes Cluster comprises numerous interconnected elements that collaborate to oversee multiple containers. These elements necessitate communication amongst themselves to execute essential functions for these containers. Moreover, containers requiring interaction with external entities require a means to establish their connections. This segment will elucidate various aspects of networking within the context of Kubernetes.

A. Container to Container

It refers to communication and interaction between two or more containers that are running within the same pod. Containers within the same pod can communicate with each other using local networking and shared storage. This means that containers within the same pod can communicate with each other using standard networking mechanisms, just as if they were running on separate virtual machines or physical servers.

When designing Kubernetes applications, it's important to consider the granularity of pods and containers based on the communication requirements. Containers within the same pod are typically tightly coupled, so this pattern is most effective when the containers need to work closely together and share resources. If containers need to communicate across pods or services, you might use Kubernetes services or other networking solutions to establish communication between them see Figure 2-11.

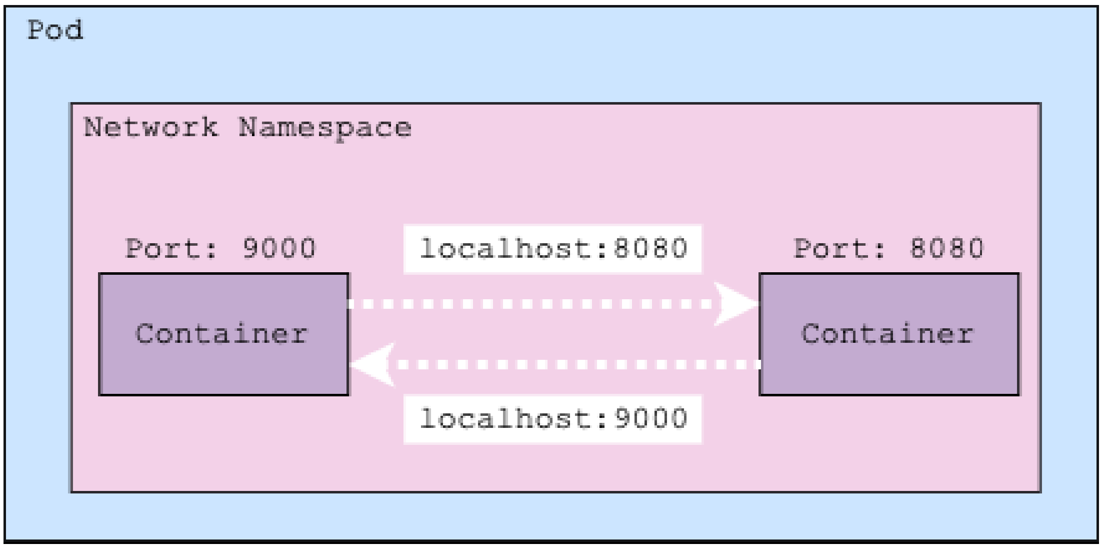


Figure 2-11 shows two containers within a single Pod establish communication with each other through the localhost interface

When containerized applications inside Pods require communication, the connection path established between them differs based on whether they reside on the same Node or different Nodes. This segment outlines how Pods interact within the same Node and across distinct Nodes.

i. **Within Same Node**

As previously mentioned, each Pod is assigned its own unique IP address, network namespace, and virtual Ethernet link known as eth0. This eth0 interface is linked to a virtual Ethernet device within the Node, referred to as vethX. This designation is accompanied by a numerical identifier, replacing the 'X,' denoting the specific Pod it is associated with—examples include veth1 and veth2- The Node's

Network Bridge, known as cbr0, employs these connections to facilitate the transmission of data packets to and from the Pods. This Network Bridge functions as a conduit that unifies two separate namespaces, effectively interconnecting all the Pods within a given Node.

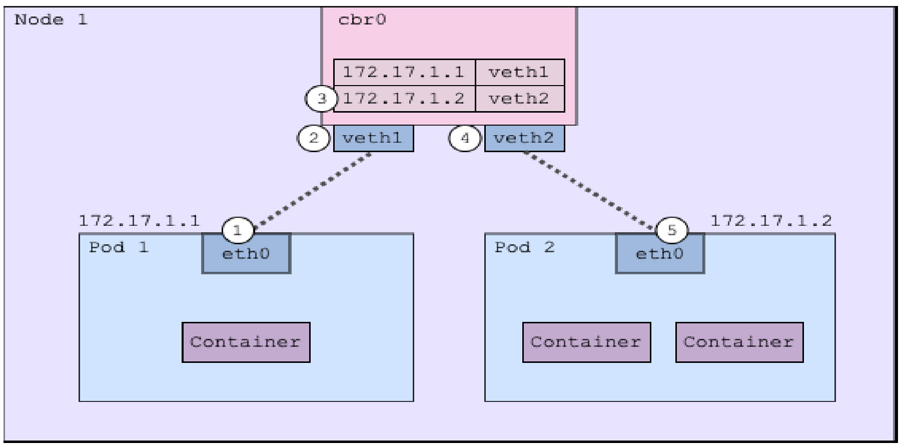


Figure 2-12 shows that Pod 1 and Pod 2 are co-located on the same Node.

Figure 2-12 illustrates the interaction between two Pods situated within the same Node. When Pod 1 intends to transmit a packet to Pod 2, familiarity with each other's IP addresses exists among all the Pods within the Node. The procedure commences with Pod 1 forwarding the packet through its eth0 interface (1), linked to veth1 (2) in the Node's namespace. Inside this namespace, the cbr0 component receives Pod 1's request and scrutinizes the connected Pods for the specified destination IP address. Upon discovery, the packet is dispatched to the designated Pod (3). In this particular instance, the packet traverses veth2 (4), which corresponds to the connection associated with Pod

2. veth2 interfaces with eth0 in Pod 2's namespace, thereby facilitating the successful delivery of the packet to its intended destination, Pod 2 (5).

ii. Between Different Nodes

Pods can establish communication with other Pods situated on different Nodes as well. As depicted in Figure 2-13, when Pod 1 located in Node 1 needs to interact with Pod 1 residing in Node 2, the same sequence of steps elucidated in the preceding section regarding Pod-to-Pod communication within the same Node is enacted. The sole distinction arises when the cbr0 of Node 1 encounters difficulty in locating the destination IP address of the target Pod. In such an instance, the request is elevated to the Cluster level (1). At this elevated tier, a routing table containing the IP address ranges assigned to each Node in the Cluster comes into play. For instance, IP addresses like 172.17.1.1 or 172.17.1.3 are attributed to all Pods in Node 1, while IP addresses akin to 172.17.2.1 or 172.17.2.3 are designated to Pods in Node 2. This routing table recognizes this pattern and designates any IP address adhering to the 172.17.2.x format as indicative of a Pod within Node 2, and likewise, IP addresses in the 172.17.1.x format are identified as Pods within Node 1 (2). Once the request has been accurately forwarded to the appropriate Node (3), the network bridge of that particular Node will discern the address of its corresponding Pod (4) and subsequently route the request to the targeted Pod, which in this case is Pod 1.

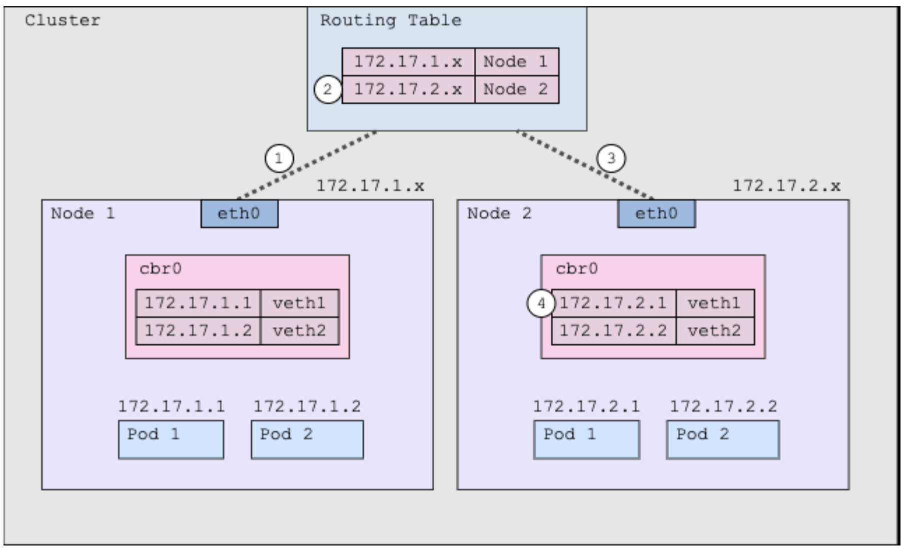


Figure 2-13 Two Pods in different Nodes communicate with each other using a routing table.

B. Services

As previously mentioned, Pods are transient elements, which implies that they frequently terminate and require reestablishment. When they are reestablished, fresh IP addresses are allocated to these new Pods, creating a challenge for other Pods to stay informed about the updated IP address associated with each individual Pod. The notion of Services offers a remedy to this issue. It entails presenting an application functioning within a Pod as a Network Service by assigning a consistent label to Pods, thereby avoiding alteration whenever a Pod must restart and its IP address changes. The execution of Services is managed by the Kube-proxy component situated in the WNs housing the Pods.

2.17.3 Calico

Calico [82] is an open-source networking and network security solution designed for containerized, virtualized, and cloud-native environments. It provides a set of tools and technologies to enable secure and scalable communication between applications and services within a Kubernetes or OpenShift cluster, as well as in other types of networked environments. Calico's role is to provide the underlying network infrastructure and capabilities that support the concept of network slicing. Network slicing involves dividing a physical network into multiple virtual networks or slices, each tailored to meet specific requirements, isolated from one another, and often associated with different services or applications. It allows to create and manage multiple logical networks (segments) within a single physical network. Each segment is isolated from others, ensuring that communication and traffic remain contained within their designated slice. In addition, it enables fine-grained network policy enforcement, allowing to define and enforce access control rules for communication between different segments. This helps ensure that traffic within each slice adheres to specific security and isolation requirements.

Calico uses the Border Gateway Protocol (BGP) to distribute routing information and manage routes between different segments. It also manages IP address allocation and assignment within each slice, ensuring that IP address ranges do not overlap between different slices.

By providing these capabilities, Calico enables the creation and management of distinct network slices with specific characteristics, policies, and requirements. This is particularly valuable in multi-tenant environments, where different user groups or applications require isolated and customized network environments, as well as in scenarios where diverse services with varying network demands coexist within the same infrastructure.

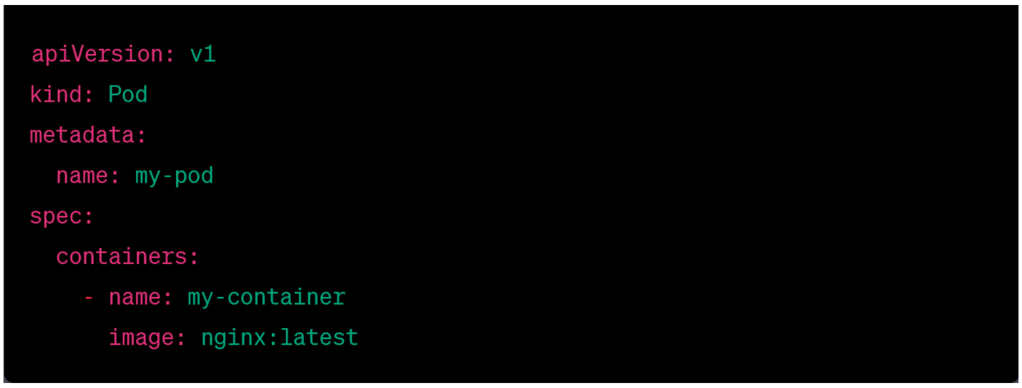
2.17.4 Prometheus

Prometheus [83] is an open-source monitoring and alerting system that is widely used in modern cloud-native environments. It is designed to collect and store time-series data from various sources, allowing users to gain insights into the performance, availability, and health of their systems. Prometheus follows a pull-based model where it periodically scrapes metrics data from instrumented targets, such as applications, services, or infrastructure components. It supports multiple data formats and protocols, including its own exposition format, and can integrate with various systems and frameworks. It stores collected metrics data in a time-series database. The database is optimized for high ingestion rates and efficient querying, enabling users to store and retain large amounts of historical data. Prometheus also provides features like data retention policies and compaction to manage the storage footprint. Prometheus offers a powerful query language called PromQL (Prometheus Query Language) for querying and manipulating metrics data. PromQL allows users to perform complex aggregations, transformations, and mathematical operations on the data. Prometheus also supports alerting based on predefined rules, which can trigger notifications when specific conditions are met. it is often used in conjunction with visualization tools like Grafana [84].

2.17.5 YAML Programming Language

In Kubernetes, YAML [85] is extensively used as a configuration language for defining and describing the various resources that make up your applications and the infrastructure they run on. YAML files are human-readable and provide a structured way to specify the desired state of your Kubernetes objects. In Kubernetes YAML can do the following:

1. **Resource Definitions**: Kubernetes resources, such as pods, services, deployments, and config maps, are defined using YAML files. Each resource has its own YAML definition that includes properties like metadata, spec, and status.



```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: my-container
      image: nginx:latest
```

Figure 2-14 YAML code example

2. **Specifying Configuration**: In YAML, you define the desired configuration and characteristics of your Kubernetes objects. This includes information such as container images, resource limits, environment variables, ports, and more.
3. **Declarative State**: Kubernetes uses the declarative approach, where you define the desired state of your application and infrastructure in YAML files. The Kubernetes control plane then

ensures that the actual state matches the desired state by making necessary adjustments.

4. **kubectl Apply**: You use the `kubectl apply` command to apply a YAML file to a Kubernetes cluster. This command reads the YAML file and creates, updates, or deletes the corresponding resources in the cluster.
5. **Managing Complex** Applications: YAML allows you to define complex applications with multiple resources and dependencies. For example, you can define a Deployment that manages a set of replica Pods, along with associated Services, ConfigMaps, and other resources.
6. **Custom Resources**: Kubernetes allows you to define Custom Resource Definitions (CRDs) using YAML, which extends Kubernetes' capabilities to suit your specific needs. CRDs enable you to create custom resources and controllers.
7. **Versioning and History**: YAML files can be version-controlled, allowing you to track changes to your Kubernetes configurations over time. You can also use tools like `kubectl rollout history` to view the history of changes to a resource.

2.17.6 DDoS Attack Generator

A DDoS generator, also known as a DDoS tool or stressor, is a type of malicious software or online service designed to launch DDoS attacks. In this study, two tools are used to generate UDP and Syn DDoS. The tools are:

A. **LOIC (Low Orbit Ion Cannon):** It is a well-recognized open-source tool for subjecting networks to stress tests. Initially designed for network stress testing, its reputation has grown due to its potential misapplication as a tool for DDoS (Distributed Denial of Service) attacks. This tool permits users to inundate a target server or website with an extensive amount of network traffic, overwhelming its capacities and leading to unresponsiveness. This is accomplished by utilizing numerous connections and concurrent requests, resulting in a distributed impact.

B. **HOIC (High Orbit Ion Cannon)**: HOIC, akin to LOIC (Low Orbit Ion Cannon), serves a comparable purpose and offers similar functionality. It operates as a network stress testing utility, originating as a proof-of-concept developed by a collective known as Anonymous. Just like LOIC, HOIC was initially conceived as a tool to evaluate the resilience of network infrastructures. However, it gained notoriety for its potential misuse as a tool for Distributed Denial of Service (DDoS) attacks. HOIC employs a strategy of utilizing numerous connections and simultaneous requests to inundate a designated server or website with an extensive influx of network traffic. This onslaught overwhelms the target's resources, potentially causing it to become unresponsive.

CHAPTER Three

Methodology

3.1 Introduction

This chapter explains the proposed system of detection and mitigation of DDoS attacks on next-generation networks based on machine learning techniques. As shown in Figure (3.1) the proposed system consists of several stages:

First stage: Generate the attack using specific tools in addition to generating legitimate packet requests, which is demonstrated in section (3.2).

Second stage: includes programming intermediate devices to treat some Pre-Prosesing cases that the authors do not believe there is an actual need to pass through the classifier to see if they are malicious or not. section (3.3)

Third stage: In this stage, how to create a Kubernetes cluster, install the network slice and the tools required for installation will be discussed, in addition to an example of how Calico is programmed to create more than one network slice. section (3.4) and section (3.5)

Fourth stage: programming the process of adding new pods to the load balancer, assigning them network addresses, and the language used to program this case will be explained. section (3.6).

Fifth stage: discussing all the details related to the machine learning language, the dataset used, and how to select the features, in addition to the basis on which one of the algorithms was chosen to represent the classifier that was used in the proposed system. section (3.7).

Sixth stage: all the steps belonging to the docker creation is discussed in section (3.8)

Seventh Stage: address the problem of dynamic allocation and how to program it, which in turn performs a set of calculations on the RAM and the CPU after sensing the consumption on it as a result of the increase in the number of attacks. section (3.8).

In the last stage, how the proposed system works will be fully discussed and a summary of all the mentioned details will be mentioned. section (3.9).

3.1.1 Infrastructure of the Proposed System

To Prevent DDoS attacks, a detection and mitigation method was implemented. To achieve this method, a development and testing environment is built in a server with the specification **CPU**: **Intel Xeon E5-2650 - Dual Processor 32-Core X 2GHz**, **Memory: 24GB DDR3 1333 ECC**. **Ubuntu Server 22.04 LTS 64-bit** is installed as an operating system. Docker was used to build the microservice, and Kubernetes was utilized as an orchestrator to manage the scaling up and scaling down processes which in turn performs the entire cluster management process and provides the best mechanism for managing resources. A metric server was developed to help the orchestrator in resource allocation and instantaneously measure the volume of usage and send data to Kubernetes to decide whether to scale up or scale down. A group of computers was used to launch the attack (Figure 3.2). Via a **Cisco switch (Version: Catalyst 2960)** PCs are connected to the server through a wired network that supports 1 Gb/s. for some filtering and assigning IP addresses to the pool of PCs, a **Cisco router (Version: 1800)** is configured. When the malicious traffic reached the microservice, through the mentioned physical network, the components of the packet were extracted from the traffic Figure 3.3, then instantaneously processed and decided whether to allow them to pass through or not. To extract data directly from the traffic, the Scapy library was used in Python code and then converted into an array and passed to the algorithm for decision-making

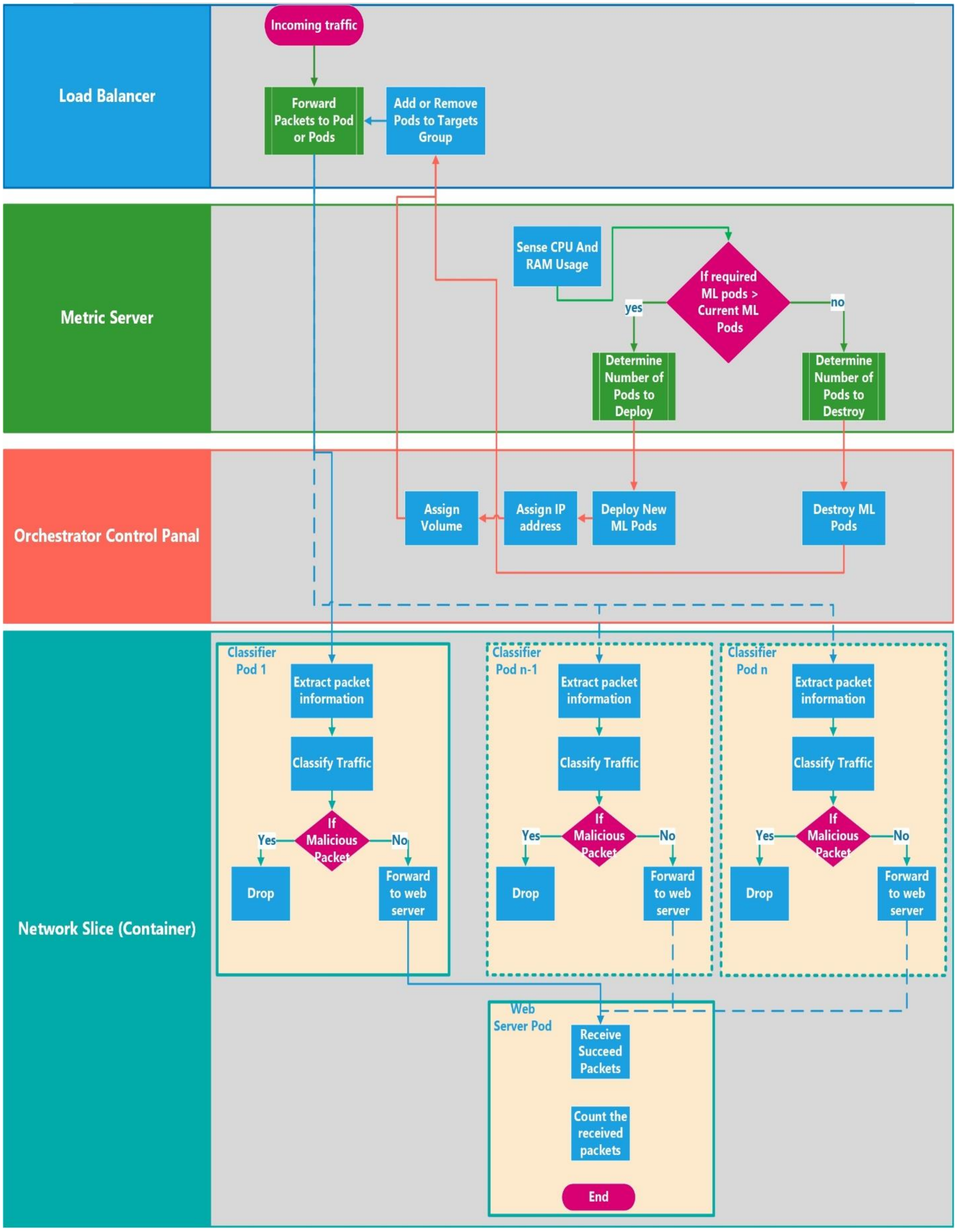


Figure 3-1 The main block diagram of the proposed system

3.2 Flood-Generating Process

To affect the virtual network resources and the communicated Pods inside it, a package of DDoS attacks was directed to the server inside the virtual network to disable it. For this purpose, **HOIC and LOIC** are used in all the attacking computers. At the same time, a simple program was added to generate legitimate **Get** requests packets to increase classification accuracy. The **Get** requests are generated using several steps described in the following steps:

```
1. Start
2. Define the target IP address and port
3. Define the relative path and query parameters for the GET request
4. Combine the IP address, port, relative path, and query parameters to form the complete URL
5. Create an HttpClient instance for sending requests
6. Repeat the following steps 10 times:
   6.1. Send a GET request to the specified URL asynchronously using HttpClient.GetAsync
   6.2. Wait for 1 second using Task.Delay
7. Wait for all the 10 requests to complete using Task.WhenAll
8. End
```

3.3 Pre-filtering Packets Before Classifying

Through the investigation around the subject of the study, packets with a source port and a destination port set to 0 are found. After searching the case, it appeared that the name of the case is "**Reserved Port Packets**". In the context of the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), port 0 is reserved and should not be used for normal data communication. Reserved Port Packets are uncommon in typical network traffic since legitimate applications and services do not use port 0 for data exchange. Such packets might be generated by certain network diagnostic tools, specialized applications, or misconfigured devices. To prevent potential security risks or unusual behavior, it is a common practice to block or filter packets with source and destination port 0 at network boundaries, such as routers and firewalls. In this study, the prevention of those packets is achieved using Access Control Lists (ACLs) of the used router to drop packets with these port values.

3.4 Set Kubernetes Cluster

Creating a Kubernetes cluster for a DDoS classification application involves setting up a cluster, deploying an application, and managing the resources. In this study, several steps have been done to create a Kubernetes cluster using kubectl [86] to deploy the DDoS classification application:

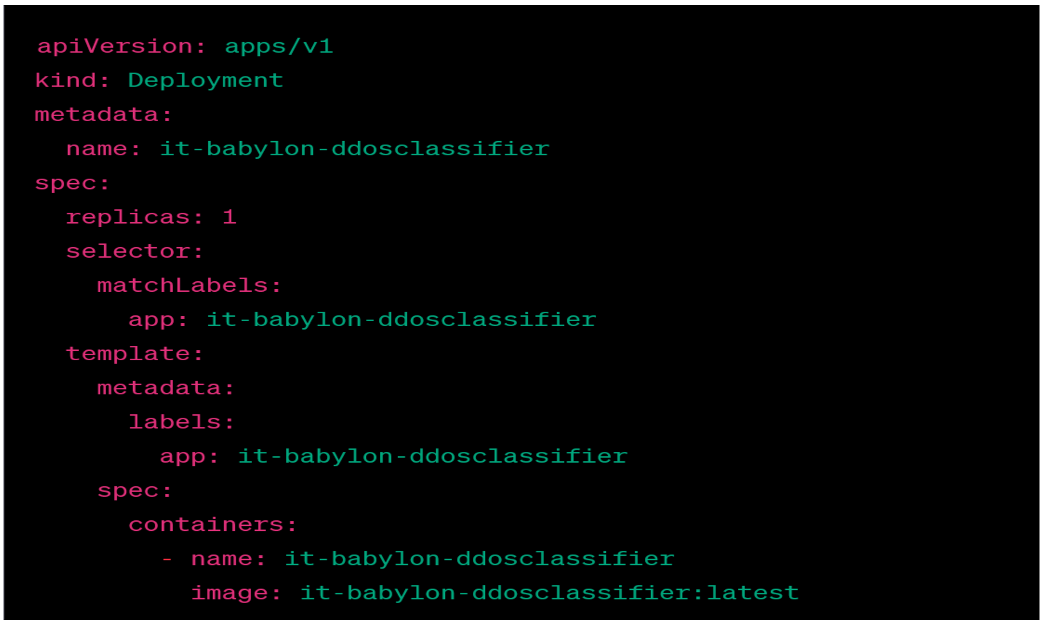
1. Install KIND: KIND stands for "Kubernetes in Docker." It is an open-source project that allows you to create local Kubernetes clusters using Docker containers as nodes. KIND provides an easy way to set up and manage lightweight Kubernetes clusters in a local development environment. It is primarily used for testing, development, and experimentation purposes. With KIND, it is able to create and destroy Kubernetes clusters to validate applications, configurations, and infrastructure changes.
2. Install kubectl: kubectl is a command-line tool that allows one to interact with Kubernetes clusters. It serves as the primary interface for managing and deploying applications, inspecting cluster

resources, and controlling Kubernetes clusters. kubectl provides a way to communicate with the Kubernetes API server, which is the control plane of a Kubernetes cluster.

3. Install a Container Runtime: Kubernetes requires a container runtime to manage and run containers. Popular options include Docker, containerd, and CRI-O. Docker is used in this study.
4. Set Up a Kubernetes Cluster: after installing all the mentioned tools, it became easy to set up a Kubernetes cluster using the following instructions:

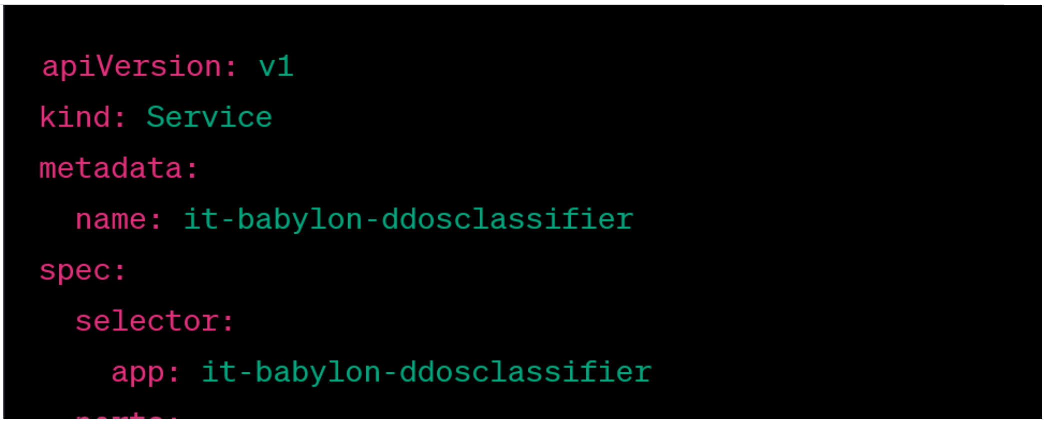
   kind create cluster --name Phd-Project

5. Create Kubernetes Deployment and Service YAML Files: Create YAML files for Kubernetes Deployment and Service resources. The Deployment manages the deployment of the application, and the Service exposes the application to the network. Figure 3.2 and Figure 3.3 illustrates the YAML file to create Kubernetes and Services.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: it-babylon-ddosclassifier
spec:
  replicas: 1
  selector:
    matchLabels:
      app: it-babylon-ddosclassifier
  template:
    metadata:
      labels:
        app: it-babylon-ddosclassifier
    spec:
      containers:
        - name: it-babylon-ddosclassifier
          image: it-babylon-ddosclassifier:latest
```

Figure 3-2 Figure illustrates the YAML file to create Kubernetes Deployment



```yaml
apiVersion: v1
kind: Service
metadata:
  name: it-babylon-ddosclassifier
spec:
  selector:
    app: it-babylon-ddosclassifier
```

Figure 3-3 illustrates the YAML file to create Kubernetes Services

6. Deploy to Kubernetes: to apply the Deployment and Service YAML files kubectl is used as the following:

kubectl apply -f deployment.yaml

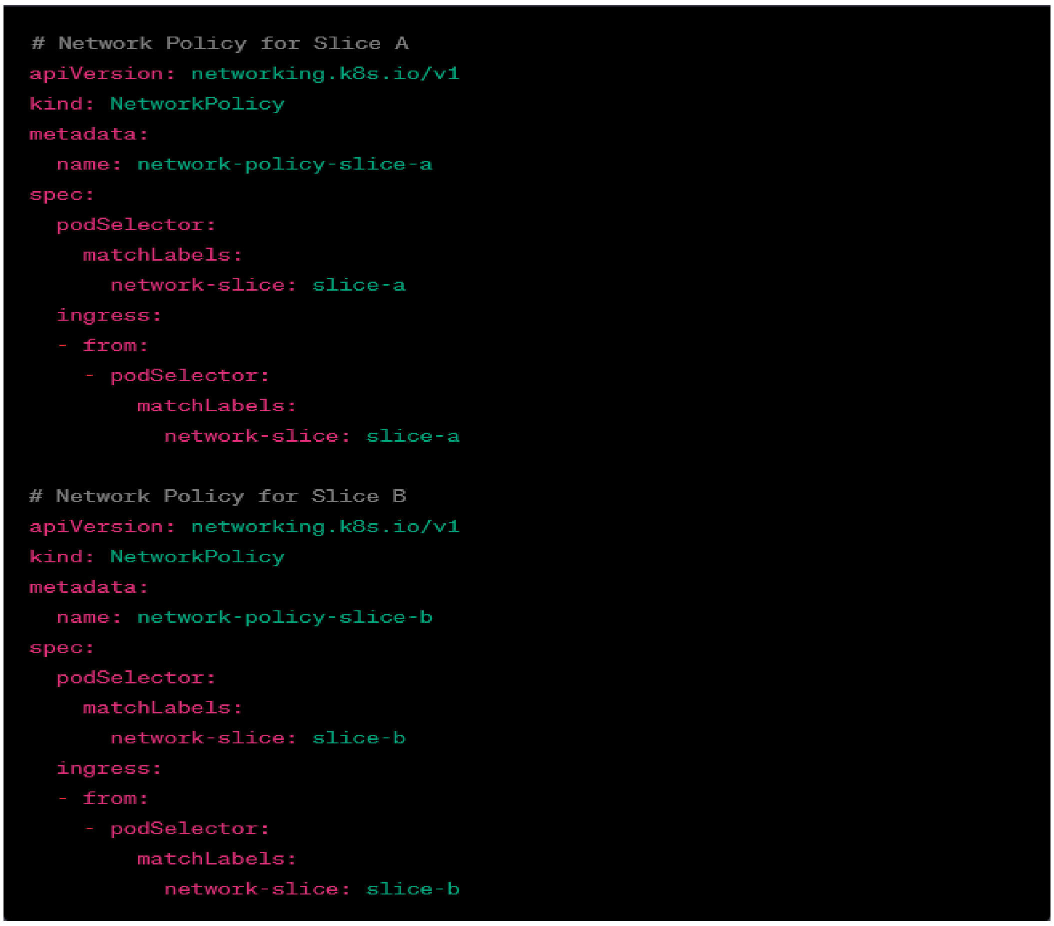
kubectl apply -f service.yaml

3.5 Installing the network slice

Creating network slices in Docker containers and Kubernetes involves using network plugins or overlays to partition and isolate the network traffic between different services or applications. Network slices are a concept commonly associated with 5G networks, where virtualized and isolated network segments are created to meet different requirements of diverse applications and services. Some third-party plugins and tools can be integrated with Kubernetes to achieve network-slicing functionality. One such popular plugin is Calico.

Anyways, to install the network slice, First, Kubernetes, Calico, and Docker need to be installed in the system. For local testing or creating a cluster **kubectl** has been used. Calico CNI (Container Network Interface) is a popular networking solution that supports network slicing and policy enforcement. In Calico, network policies define the isolation and communication rules between different network slices. Each network policy defines the rules for incoming and outgoing traffic for pods labeled with specific labels.

To create Calico network policies to enforce isolation between slices, the YAML code was used. See Figure 3.4



```yaml
# Network Policy for Slice A
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: network-policy-slice-a
spec:
  podSelector:
    matchLabels:
      network-slice: slice-a
  ingress:
  - from:
    - podSelector:
        matchLabels:
          network-slice: slice-a

# Network Policy for Slice B
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: network-policy-slice-b
spec:
  podSelector:
    matchLabels:
      network-slice: slice-b
  ingress:
  - from:
    - podSelector:
        matchLabels:
          network-slice: slice-b
```

Figure 3-4 a capture of Calico configuration to create two network slices

3.6 Programming the process of adding new pods to the target group

In Kubernetes, Pods are dynamic and can have ephemeral IP addresses that may change when they are recreated or rescheduled. To handle this situation, an Ingress Controller with an external load balancer to manage the routing of traffic to the Pods is used. Ingress Controllers are responsible for handling incoming traffic and routing it to the appropriate services or Pods based on the rules defined in the Ingress resource.

To add a new Pod IP address to the targets group in the load balancer, YAML language is used to build a file with the name Upstream. The file contains the IP addresses and port numbers (socket) for all the target groups. after the deployment of the new Pod, the identified Pod IP address and port number should be added to the targets group in the Upstream file.

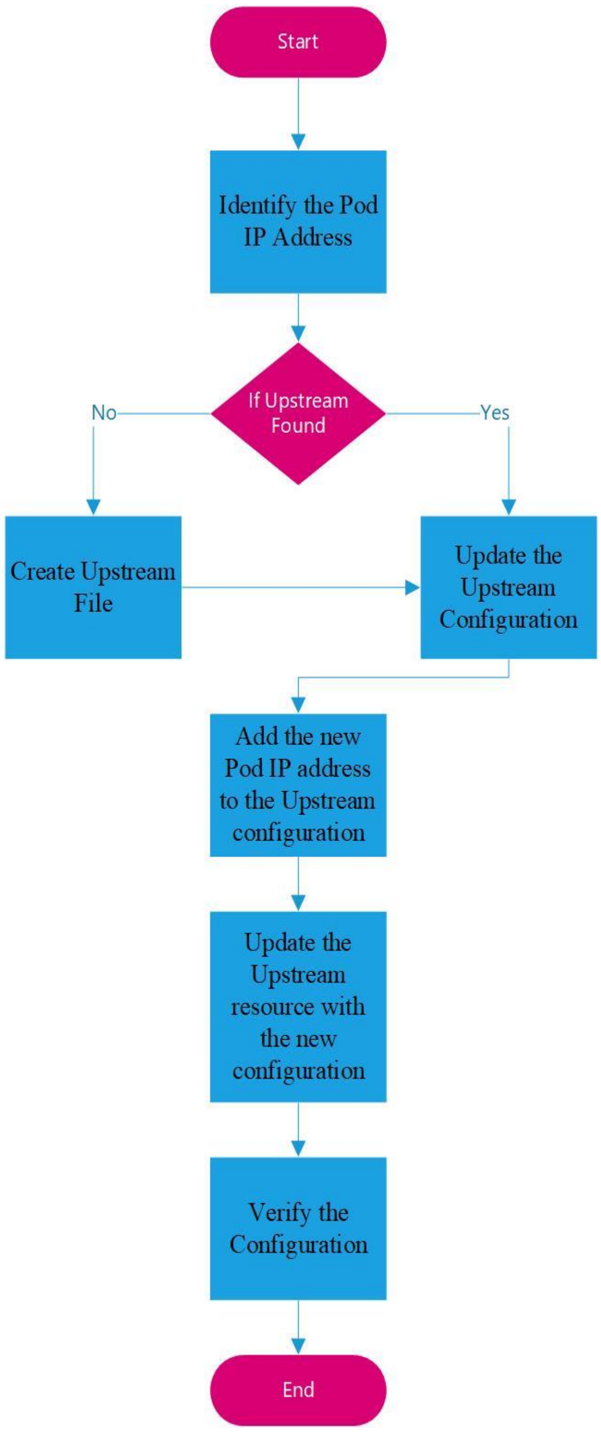

Start
Identify the Pod IP Address
If Upstream Found
No
Yes
Create Upstream File
Update the Upstream Configuration
Add the new Pod IP address to the Upstream configuration
Update the Upstream resource with the new configuration
Verify the Configuration
End

Figure 3-5 flowchart of adding new Pods to load balancer Targets group

The process of adding the Pod IP address is done as the flowchart in Figure 3.5:

3.7 ML-Based Detection Approach

There are many ML techniques applicable to DDoS detection. This study applies 4 machine learning classifiers to perform the multi-classification task for identifying the unknown DDoS traffic. This section and its subsections explain the main techniques of the ML stage, such as dataset, preprocessing, feature selection, and how the suitable classifier has been chosen.

3.7.1 Datasets

The CICDoS2019 dataset is a comprehensive dataset that has been developed by the Canadian Institute for Cybersecurity (CIC) [5]. It consists of millions of records of both legitimate and malicious traffic.

The data was collected from the Canadian Institutes of Cybersecurity (CIC) honeypot network during the period from August to October 2019. The data set is designed to enable the development of new machine learning algorithms that can accurately detect and classify DDoS attacks. The data set is divided into two parts, namely the benign traffic and the malicious traffic. The benign traffic consists of normal user-initiated activities such as web browsing, email, file transfers, etc. The malicious traffic includes various types of network-based attacks such as TCP SYN floods, UDP floods, ICMP floods, etc see Table (3.1). All the traffic is labeled according to its type. For example, all the malicious traffic is labeled as ‘malicious’ while all the benign traffic is labeled as ‘benign’. The CICDDoS2019 dataset also contains several features that can be used to characterize the traffic. These features include the source and destination IP addresses, ports, packet length, payload size, protocol, flags, etc. All the records in the dataset also contain the timestamp of when the traffic was generated. This helps in understanding the timing of the attack and can be used for further analysis. The CICDDoS2019 dataset also contains a large number of attack types and patterns. Overall, the CICDDoS2019 dataset provides an excellent platform for researchers and practitioners to develop new machine-learning algorithms for accurately detecting and classifying DDoS attacks. It is a great resource for the development of new machine-learning algorithms that can accurately detect and classify DDoS attacks.

Table 3-1: CICDDoS-2019 dataset summary

Dataset DDos Attack Files	Label	Quantity	Ratio Percentage	The Total Number
LDAP	BENIGN	1602	0.07	2,181,530
	DDoS_LDAP	2,179,928	99.93	
MSSQL	BENIGN	1995	0.04	4,524,484
	DDoS_MSSQL	4,522,489	99.96	
DNS	BENIGN	3380	0.07	5,074,382
	DDoS_DNS	5,071,002	99.93	
NetBIOS	BENIGN	1705	0.04	4,094,978
	DDoS_NetBIOS	4,093,273	99.96	
NTP	BENIGN	14,337	1.18	1,216,976
	DDoS_NTP	1,202,639	98.82	
UDP	BENIGN	2151	0.07	3,136,794
	DDoS_UDP	3,134,643	99.93	
SNMP	BENIGN	1502	0.03	5,161,365
	DDoS_SNMP	5,159,863	99.97	
SSDP	BENIGN	762	0.03	2,611,372
	DDoS_SSDP	2,610,610	99.97	
SYN	BENIGN	389	0.03	1,380,404
	DDoS _Syn	1,380,015	99.97	

Over the course of working days, network traffic flows were recorded, and 84 features were retrieved as shown in Table (3.2). In addition, the CICIDS2017, NF-BoT-IoT-v2 dataset has been used to validate the proposed classifier.

Table 3-2: The features of the CICIDS2019 Dataset

No	Feature Name	No	Feature Name	No	Feature Name
1	Flow ID	29	Fwd IAT Std	57	ECE Flag Count
2	Source IP	30	Fwd IAT Max	58	Down/Up Ratio
3	Source Port	31	Fwd IAT Min	59	Average Packet Size
4	Destination IP	32	Bwd IAT Total	60	Avg Fwd Segment Size
5	Destination Port	33	Bwd IAT Mean	61	Avg Bwd Segment Size
6	Protocol	34	Bwd IAT Std	62	Fwd Avg Bytes/Bulk
7	Timestamp	35	Bwd IAT Max	63	Fwd Avg Packets/Bulk
8	Flow Duration	36	Bwd IAT Min	64	Fwd Avg Bulk Rate
9	Total Fwd Packets	37	Fwd PSH Flags	65	Bwd Avg Bytes/Bulk
10	Total Backward Packets	38	Bwd PSH Flags	66	Bwd Avg Packets/Bulk
11	Total Length of Fwd Packets	39	Fwd URG Flags	67	Bwd Avg Bulk Rate
12	Total Length of Bwd Packets	40	Bwd URG Flags	68	Subflow Fwd Packets
13	Fwd Packet Length Max	41	Fwd Header Length	69	Subflow Fwd Bytes
14	Fwd Packet Length Min	42	Bwd Header Length	70	Subflow Bwd Packets
15	Fwd Packet Length Mean	43	Fwd Packets/s	71	Subflow Bwd Bytes
16	Fwd Packet Length Std	44	Bwd Packets/s	72	Init_Win_bytes_forward
17	Bwd Packet Length Max	45	Min Packet Length	73	Init_Win_bytes_backward
18	Bwd Packet Length Min	46	Max Packet Length	74	act_data_pkt_fwd
19	Bwd Packet Length Mean	47	Packet Length Mean	75	min_seg_size_forward
20	Bwd Packet Length Std	48	Packet Length Std	76	Active Mean
21	Flow Bytes/s	49	Packet Length Variance	77	Active Std
22	Flow Packets/s	50	FIN Flag Count	78	Active Max
23	Flow IAT Mean	51	SYN Flag Count	79	Active Min

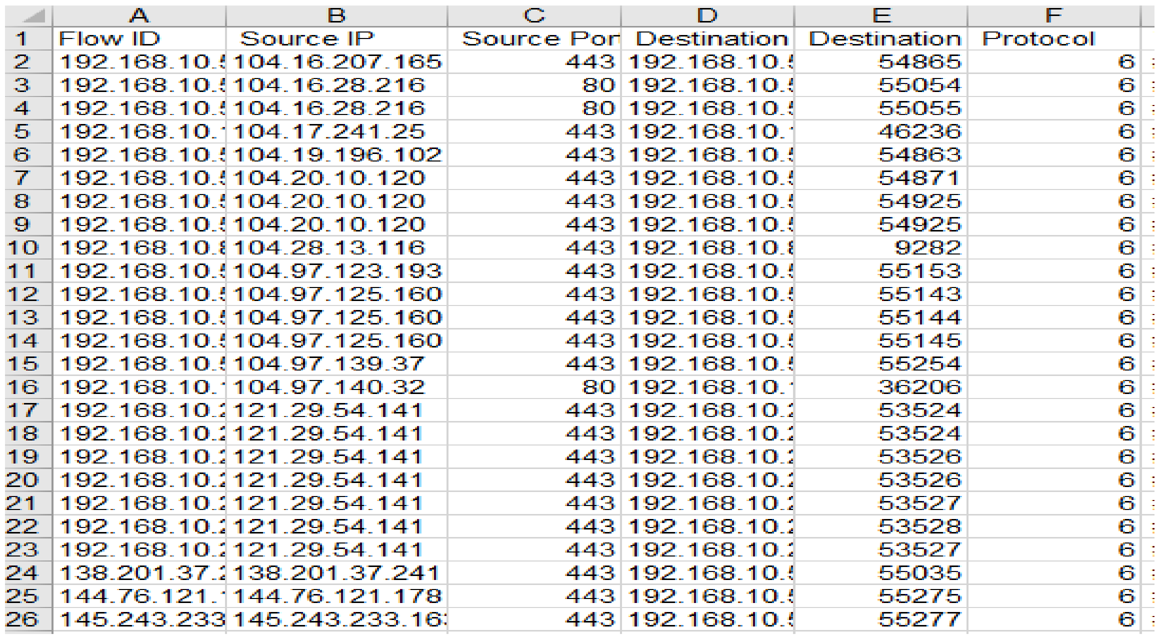
No	Feature Name	No	Feature Name	No	Feature Name
24	Flow IAT Std	52	RST Flag Count	80	Idle Mean
25	Flow IAT Max	53	PSH Flag Count	81	Idle Std
26	Flow IAT Min	54	ACK Flag Count	82	Idle Max
27	Fwd IAT Total	55	URG Flag Count	83	Idle Min
28	Fwd IAT Mean	56	CWE Flag Count	84	Label

3.7.2 Preprocessing

The initial download of the first CICDDoS2019 dataset included all its features. To ensure data cleanliness, any instances of Not a Number (NaN) values and duplicate columns were removed. Within this process, a redundancy was identified in the "Fwd Header Length" feature, leading to the elimination of one of them. The dataset was then narrowed down to utilize only 320,000 records from four different files. Among these records, there were 64,000 instances for each type of DDoS attack (UDP, SYN, Portmap, MSSQL) and an additional 64,000 benign records.

It is important to note that special attention was paid to addressing flawed data within the dataset. For instance, records with negative values were excluded from the dataset. Furthermore, all records with a source port or destination port value of zero were also eliminated.

To enhance the precision of outcomes, feature selection was carried out based on widely recognized criteria. A visual representation of a subset of the CICDDoS2019 dataset can be found in Figure (3.6).



	A	B	C	D	E	F
1	Flow ID	Source IP	Source Por	Destination	Destination	Protocol
2	192.168.10.	104.16.207.165	443	192.168.10.	54865	6
3	192.168.10.	104.16.28.216	80	192.168.10.	55054	6
4	192.168.10.	104.16.28.216	80	192.168.10.	55055	6
5	192.168.10.	104.17.241.25	443	192.168.10.	46236	6
6	192.168.10.	104.19.196.102	443	192.168.10.	54863	6
7	192.168.10.	104.20.10.120	443	192.168.10.	54871	6
8	192.168.10.	104.20.10.120	443	192.168.10.	54925	6
9	192.168.10.	104.20.10.120	443	192.168.10.	54925	6
10	192.168.10.	104.28.13.116	443	192.168.10.	9282	6
11	192.168.10.	104.97.123.193	443	192.168.10.	55153	6
12	192.168.10.	104.97.125.160	443	192.168.10.	55143	6
13	192.168.10.	104.97.125.160	443	192.168.10.	55144	6
14	192.168.10.	104.97.125.160	443	192.168.10.	55145	6
15	192.168.10.	104.97.139.37	443	192.168.10.	55254	6
16	192.168.10.	104.97.140.32	80	192.168.10.	36206	6
17	192.168.10.	121.29.54.141	443	192.168.10.	53524	6
18	192.168.10.	121.29.54.141	443	192.168.10.	53524	6
19	192.168.10.	121.29.54.141	443	192.168.10.	53526	6
20	192.168.10.	121.29.54.141	443	192.168.10.	53526	6
21	192.168.10.	121.29.54.141	443	192.168.10.	53527	6
22	192.168.10.	121.29.54.141	443	192.168.10.	53528	6
23	192.168.10.	121.29.54.141	443	192.168.10.	53527	6
24	138.201.37.	138.201.37.241	443	192.168.10.	55035	6
25	144.76.121.	144.76.121.178	443	192.168.10.	55275	6
26	145.243.233	145.243.233.16	443	192.168.10.	55277	6

Figure 3-6 Sample of CICDDoS2019 dataset

3.7.3 Feature Selection Methods

Feature selection methods play an important role in improving model performance, there is a tradeoff between the number of features and the time complexity of an algorithm, the more features model leads to the most time complexity and the most accuracy, and vice versa. The number of features in the data set is 84. CICDDoS2019 is rich in features as compared to the rest of the datasets with a small number of features like the UCAL dataset [87]. Therefore, it is important to use feature selection techniques to reduce the number of features to enhance the current study.

The features were chosen using a wrapper feature selection method and using information gain to reduce the number of features by ranking the features depending on the required number of features. At first, The wrapper feature selection selects 12 features then an information gain is used to create 8 groups with a different number of features but all the features belong to the wrapper group.

The proposed ML model dataflow is depicted in Figure (3.7). Preprocessing of the training data will be done initially in the offline mode before running the proposed system. Algorithm (3.2) and (3.3) is used to select relevant features and eliminate worthless or not applicable features.

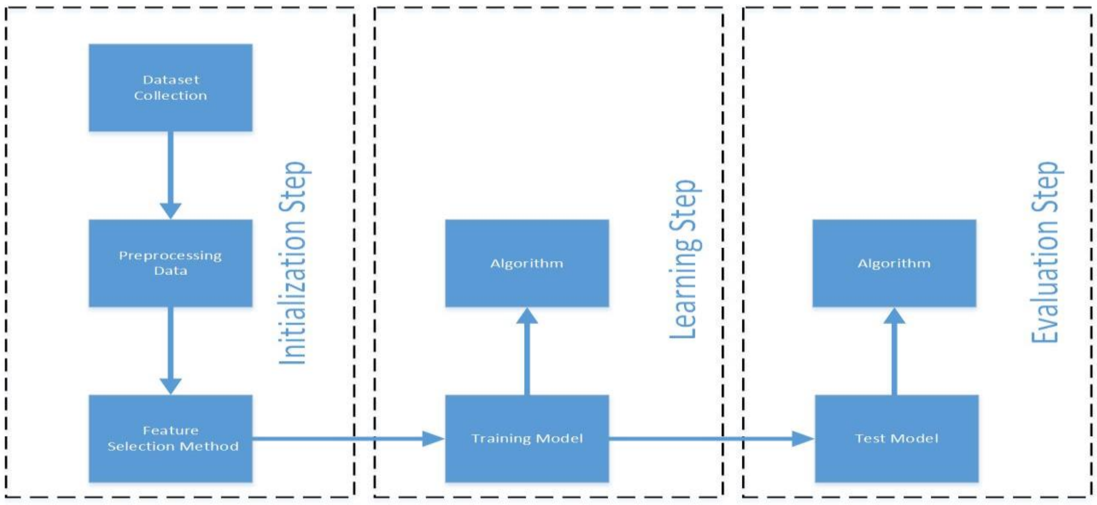


Figure 3-7 The ML and features selection process

Algorithms (3.2, 3.3) are utilized for selecting the most appropriate feature that distinguishes irregular traffic from normal traffic. The stage of selecting and refining features is of utmost importance within the realm of machine learning. Not all features within a dataset are employed by machine learning algorithms. Employing the complete set of features for constructing a predictive model is not only resource-intensive, but it also prolongs the process. Initially, data must undergo normalization, deduplication, or rectification to address imbalanced data. In practical scenarios, datasets such as DDoS attack traffic tend to be extensive. Therefore, to effectively train these vast datasets containing numerous features, it is imperative to eliminate irrelevant or redundant information from the dataset. Moreover, insignificant or redundant features may negatively impact the performance of the detection mechanism. The distribution of attack classes in the CICDDoS2019 dataset tends to be uneven. Additionally, the datasets encompass attributes that lack relevance when it comes to identifying attacks.

Consequently, an adopted approach for feature selection is employed to identify extraneous features among the initial set of 84 features, serving as the initial phase within the machine learning process, as elaborated upon in Chapter 4.

Algorithm (3.1): Wrapper algorithm

```
Input: F = Training dataset, processing n features f1, f2, f3... fn
Output: The selected features list
    1.  Initialize an empty set S to hold the selected features.
    2.  Initialize a set F with all the features in the dataset.
    3.  While F is not empty, do the following:
            3.1. For each feature f in F, add f to S, and build a J48
                 decision tree using S as the feature set.
                    3.1.1. Evaluate the performance of the decision tree.
                    3.1.2. Keep track of the performance for each feature
                           and decision tree.
                    3.1.3. Remove the feature that resulted in the highest
                           performance from S, if S is not empty.
                    3.1.4. If removing the feature decreased performance or
                           if S is empty, stop and return the set S as the selected
                           features.
            3.2 End for
    4.  End while
    5. End
```

Algorithm (3.2): Filter Feature Selection

```
Input: F: Dataset with features (X) and target variable (y).
Output: A selected subset of features
1. Calculate Feature Scores:
    a. For each feature in X:
        i. Apply a statistical test or compute a correlation
           score between the feature and the target variable.
        ii. Assign a score to each feature based on the test or
            correlation value.
2. Rank Features:
    a. Sort features in descending order based on their scores.
3. Select Top Features:
    a. Choose the top-k features with the highest scores.
4. Return Selected Features.
5. End
```

3.7.4 Classifier Selection

To utilizing the dataset, a classification models developed to employing diverse machine learning techniques following the identification of optimal feature subsets through algorithms 3.2 and 3.3. Leveraging multiple learning models such as MP, REPT, PART, RF, and J48 - widely recognized supervised learning algorithms - their performance are gauged. Furthermore, the predicted results are enhanced by integrating the best two classifiers as a stacked classifier.

The output of the approach outlined in algorithm 3.4 yields the most effective classifier, which will be integrated to build the new stacked classifier. This integrated classifier will subsequently be deployed as a real-time microservice application. This application serves the purpose of detecting DDoS attacks within the context of a next-generation network environment.

Algorithm (3.3): ClassifierSelect

```
Input: FL = Features_List
Output: Features Subsets, Accuracy, and Testing time with Fast and
        Accurate Model
1.  Begin
2.   For every feature Fr in Feature_Ranked data
3.     Start to Select from Feature Sets
4.       SET1/Groups1 features
5.       SET2/Groups2 features
6.       SET3/Groups3 features
7.       SET4/Groups4 features
8.       SET4/Groups5 features
9.       SET4/Groups6 features
10.      SET4/Groups7 features
11.      SET4/Groups8 features
12.      SET4/Groups9 features
13.       For each Feature in SETs/Groups
14.       Feed Selected features to MP, REPT, PART, RF, J48,
    Stacked_Modle
15.          Apply Classifier
16.             C1 = MultilayerPerseptron model
17.             C2 = REP Tree model
```

```
18.            C3 = PART model
19.            C4 = Random Forest model
20.            C5 = J48 model
21.            C6= Stacked_Modle
22.         Calculate Test time, Tree size, and Accuracy
23.         Compare the Accuracy and testing time of C1, C2, C3,
    C4, and C5
24.END Algorithm
```

During the set of experiments that were conducted, various machine learning algorithms were evaluated to determine the most suitable one for building the microservices responsible for the detection and prevention of network attacks. The evaluation criteria included both the accuracy of results and the speed of the training phase. The dataset comprised a diverse set of network packets, including both normal and malicious traffic, to provide a realistic representation of real-world network environments. The accuracy of results was assessed by measuring the algorithm's ability to correctly classify network traffic as normal or malicious. A high true positive rate and a low false positive rate were desirable to minimize both missed detections (false negatives) and false alarms (false positives). Furthermore, the training phase's speed was evaluated to ensure that the chosen algorithm could efficiently process large-scale network traffic data without excessive computational overhead. After careful analysis and comparison of the experimental results, it was found that the stacked model which combines RF and J48 algorithm, demonstrated outstanding performance. It achieved the highest accuracy in classifying network traffic with a true positive rate and a false positive rate. Moreover, the training phase of the model was significantly faster compared to other complex algorithms.

Based on these findings, the stacked model algorithm was deemed the most suitable choice for building the microservices responsible for network attack detection and prevention. Its high accuracy ensures reliable threat detection, while its fast training phase allows for real-time or near real-time processing of network traffic data. The adoption of the stacked model algorithm in the microservices architecture will provide a robust and efficient solution to safeguard the network infrastructure from potential threats.

3.7.5 Stacked Model

A stacked ensemble or stacked model, is a machine learning technique that combines the predictions of multiple base models (or learners) to create a more robust and accurate final prediction. The concept behind a stacked classifier is to leverage the strengths of different base models by allowing them to "vote" or contribute their predictions for a given input, and then combining these predictions to make a final decision.

In this experiment, two algorithms RF and J48 have been used as a stacked model Algorithm (3.5) to accomplish the best results during the experiment, It's worth noting that building and tuning stacked classifiers can be more complex and computationally intensive compared to using a single model. However, when done correctly, they can lead to significant improvements in predictive performance.

Algorithm (3.4): stacked model algorithm

```
Input: Training dataset, Validation dataset, Test dataset, List of base
models, Metamodel.
Output: Final classification for the test dataset
Begin
#Training Base Models
    1.  For each base_model in base_models:
            a.  Train base_model on X_train and y_train
    2.  End For
#Generating Base Model Predictions
    3.  Create empty arrays for base_model_classification_train and
        base_model_ classification _val
    4.  For each base_model in base_models:
            a.  Classify base_model on X_train and X_val
            b.  Append the classification to base_model_ classification _train
                and base_model_ classification _val
    5.  End For
#Training the Meta Model
    6.  Train the meta_model on base_model_ classification _train and
        y_train
    7.  Generating Meta Model Input
    8.  Classify each base_model on X_test to generate base_model_
        classification _test
    9.  Making Final Predictions
    10. Classify meta_model on base_model_ classification _test to get the
        final classification for the test dataset
    11. End
```

3.7.6 ML Performance

During this phase, metric measurements are employed to demonstrate the impact across five different classifiers. The evaluation of classifier performance involved utilizing four primary performance metrics. These metrics encompass Fit Time, Accuracy, Recall, and Precision. Accuracy signifies the algorithm's precision in distinguishing attacks across both normal and attack-related traffic. Recall captures the ability to accurately detect genuine attack instances within the incoming assault traffic. Precision pertains to identifying an attack amidst the instances predicted as positive cases.

Furthermore, alongside the aforementioned four evaluation criteria, Fit Time is also computed. Fit Time denotes the duration taken by the classifier to fit during the testing phase.

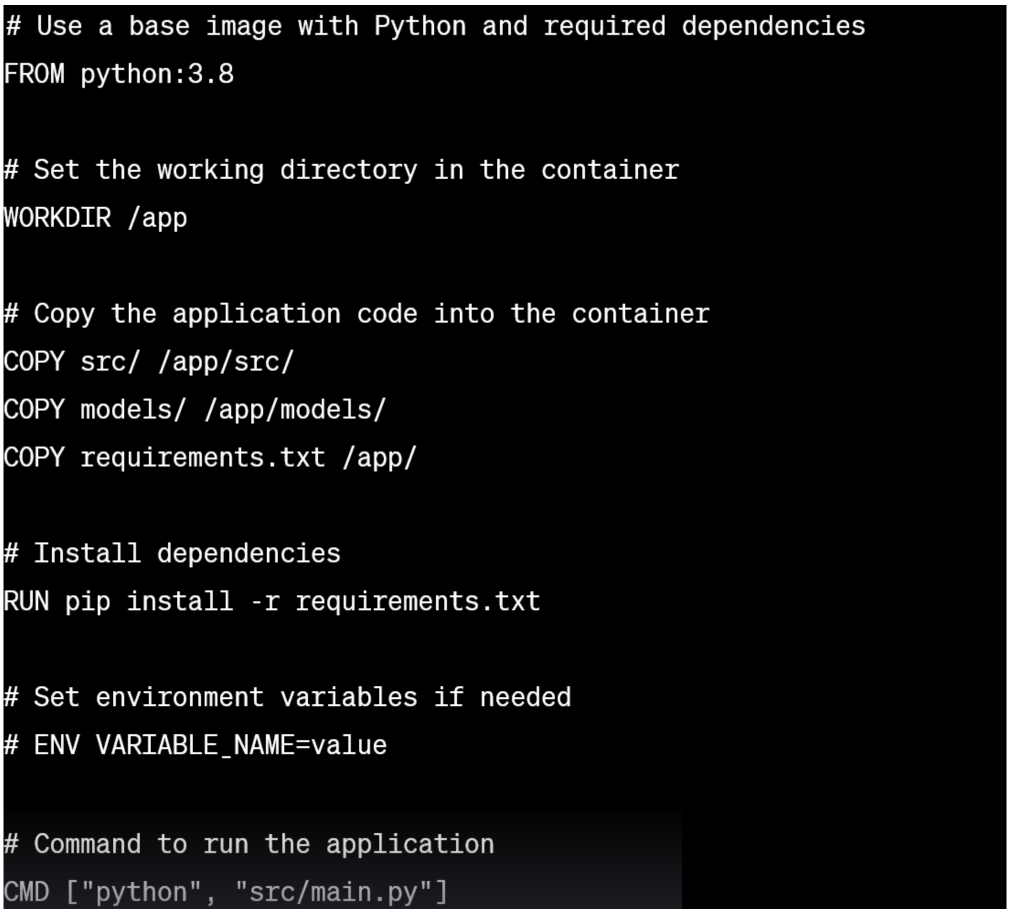
3.8 Creating a Docker image for a DDoS classification

Creating a Docker image for a DDoS classification application involves packaging the application, its dependencies, and any required configuration into a Docker container. In this study a Docker image for a DDoS classification application is built using the following steps:

1. Create Application Files: necessary files for a DDoS classification application are created including source code, trained machine learning models, and some other configuration files. Python was used to create the mentioned requirement

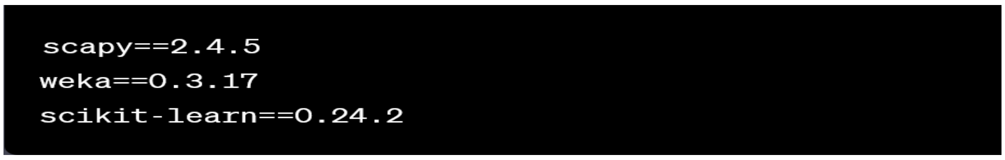
2. Create a Dockerfile: Create a file in the root directory of the application. This file defined the steps needed to build the Docker

image. Figure 3.8 and Figure 3.9 shows a Dockerfile and requirement file used in this study.



```
# Use a base image with Python and required dependencies
FROM python:3.8

# Set the working directory in the container
WORKDIR /app

# Copy the application code into the container
COPY src/ /app/src/
COPY models/ /app/models/
COPY requirements.txt /app/

# Install dependencies
RUN pip install -r requirements.txt

# Set environment variables if needed
# ENV VARIABLE_NAME=value

# Command to run the application
CMD ["python", "src/main.py"]
```

Figure 3-8 illustrates one of the docker files that have been used in creating the classifiers



```
scapy==2.4.5
weka==0.3.17
scikit-learn==0.24.2
```

Figure 3-9 The content of the requirement file which mentioned in the Dockerfile

3. Build the Docker Image: the docker image is built by executing the following code:

docker build -t it-babylon-ddosclassifier

3.9 Programming the metric server

The Metric Server is a file with a configuration that collects and provides resource utilization metrics for the nodes and pods in a cluster. It enables the Horizontal Pod Autoscaler (HPA) and other components to dynamically allocate CPU and RAM resources based on the actual resource utilization of the pods. In a Kubernetes cluster, each node runs a set of pods, and these pods require CPU and RAM resources to perform their tasks. The amount of CPU and RAM allocated to each pod can significantly impact the application's performance and overall cluster efficiency. Allocating more resources than needed can lead to resource wastage while allocating fewer resources can cause performance bottlenecks.

The Metric Server addresses these issues by continuously monitoring the resource utilization of nodes and pods and providing this information to other Kubernetes components, such as the Horizontal Pod Autoscaler.

In the proposed system, The Metric Server collects resource utilization metrics from the Kubernetes nodes and the pods running on those nodes. The metrics collected include CPU and memory usage, as well as other relevant metrics such as network utilization. The collected metrics are aggregated and stored for efficient querying. The Metric Server uses in-memory storage for recent metrics and allows querying of resource utilization data over a specific time window.

The Metric Server exposes an API that other Kubernetes components can use to query resource utilization metrics. This API is designed to be lightweight and allows fast access to the most recent metrics data. it is an essential component for Horizontal Pod Autoscaler (HPA) functionality in Kubernetes. The HPA uses the metrics provided by the Metric Server to automatically adjust the number of replicas for a deployment or replica set based on CPU and memory utilization metrics. With the HPA configured and the Metric Server providing real-time resource utilization data, Kubernetes can automatically scale the number of replicas up or down based on the actual demand for CPU and memory resources. This dynamic resource allocation ensures that applications get the resources they need to perform optimally without over-allocating resources and wasting cluster capacity.

In this experiment, the metric server decides how many Pods should be run at any moment depending on the following equation:

- $\mathrm{R} = \frac{\Upsilon}{\mathrm{r}} + \delta$ --3.1

Where R is the required number of Pods (the range of R id from (1 to the number of server CPU cores), Υ is the current CPU usage in milli core mC, r is a constant (1000 mC) = 1 Core, and $\delta = \begin{cases} 0, & \Upsilon \text{ mode r} = 0 \\ 1, & \Upsilon \text{ mode r} > 0 \end{cases}$.
Regarding the value of R in equation 3.1, the value of T (total number of running pods) can be calculated by $T = \begin{cases} R - C, & \text{if R} > \text{C} \\ R, & if\ R \leq C \end{cases}$ where C is the current running number of pods.
Finally, the value of the T is assigned to C when the metric server remeasures the CPU usage after 15 seconds and recalculates the Total number of pods T.

3.10 Test of the Proposed Work

After the attack is launched from the physical PCs, the network traffic will be directed to the network slice in the server via the switch, the router, and the IP address of the network slice. Inside the network slice the load balancer will forward the incoming packet to the classifiers depending on the mechanism of the least connection.

When the classifier receives the packets, it will classify them as malicious or not based on the ML algorithm rules created in the training stage. If it is malicious, then drop it, otherwise keep forwarding it to the web server inside the same slice.

At the same time, the metric server keeps sensing the usage of CPU and RAM, if the consumption of the Pod exceeds 1000 mC then deploy another Pod, and destroy the Pod if the CPU consumption requires less than the current Pods.

CHAPTER Four

Result Discussion and Analysis

4.1 Introduction

This chapter demonstrates the results of the proposal for detection and mitigation methods for DDoS attacks. The chapter also discusses the results of the proposed work and explains the methodology presented in chapter three.

4.2 ML-based DDoS Attacks Detection

One of the highly relentless attacks is the crucial distributed DoS attacks. The types and tools for these attacks increase day-to-day as the technology increases. So the methodology for detection of DDoS should be advanced. For this purpose, an automated DDoS proposed system is created using ML which can run on any commodity hardware. Several ML algorithms and datasets which will discussed in the following are used to achieve the targeted goal.

4.2.1 Datasets

As mentioned in Chapter 3, the CICDDoS2019 dataset will be used for building ML models and testing them. CICIDS2017 will also be used to verify the proposed model. Since the two datasets do not differ much in features, the first dataset features will be reviewed in this section.

4.2.2 Feature Selection Methods

Feature selection is essential for removing unnecessary features that increase the time complexity of the model during testing and may lead to model failure. This section presents two dependent mechanisms of feature selection as will be demonstrated in the next sections. According to Letritures [19] – [25], many experiments have been done using different methods but the most efficient method was when combining **Wrapper** and **Information gain.** Two terms (GROUPS and SETS) will be used to describe each outcome referred to as multiple features of the two mechanisms of feature selection.

A. The Wrapper Feature Selection

Wrapper feature selection is a technique used in machine learning and feature engineering to improve the performance of a model by selecting a subset of relevant features from the original set of features. Instead of using all available features, wrapper methods involve repeatedly training and evaluating the model using different subsets of features to find the best combination that optimizes the model's performance.

Table (4.1) shows the features that have been achieved using the Wrapper method after implementing it on the CICDDoS2019 dataset.

Table 4-1 features returned by wrapper method

#	**Feature**
1.	Destination Port
2.	Protocol
3.	Total Length of Fwd Packets
4.	Fwd IAT Std
5.	Min Packet Length
6.	Packet Length Std
7.	RST Flag Count
8.	Source Port
9.	Init_Win_bytes_forward
10.	Init_Win_bytes_backward
11.	act_data_pkt_fwd
12.	Packet Length Mean

B. The Filter Base Feature Selection

To reduce the number of features in Table 4-1, a filter-based feature selection using information gain is implemented. The process of filtering the features according to their score is repeated several times to get 8 sets of features in addition to Wrapper set of features as illustrated in Table 4-2 to Table 4-10

Table 4-2 same features of wrapper returned features

#	Feature
1.	Destination Port
2.	Protocol
3.	Total Length of Fwd Packets
4.	Fwd IAT Std
5.	Min Packet Length
6.	Packet Length Std
7.	RST Flag Count
8.	Source Port
9.	Init_Win_bytes_forward
10.	Init_Win_bytes_backward
11.	act_data_pkt_fwd
12.	Packet Length Mean

Table 4-3 features return by first filtering using information gain

#	Feature
1.	Destination Port
2.	Protocol
3.	Total Length of Fwd Packets
4.	Fwd IAT Std
5.	Min Packet Length
6.	Packet Length Std
7.	Source Port
8.	Init_Win_bytes_forward
9.	Init_Win_bytes_backward
10.	act_data_pkt_fwd
11.	Packet Length Mean

Table 4-4 features return by second filtering using information gain

#	Feature
1.	Destination Port
2.	Protocol
3.	Total Length of Fwd Packets
4.	Fwd IAT Std
5.	Min Packet Length
6.	Packet Length Std
7.	Source Port
8.	Init_Win_bytes_forward
9.	act_data_pkt_fwd
10.	Packet Length Mean

Table 4-5 features return by third filtering using information gain

#	Feature
1.	Destination Port
2.	Protocol
3.	Total Length of Fwd Packets
4.	Min Packet Length
5.	Packet Length Std
6.	Source Port
7.	Init_Win_bytes_forward
8.	act_data_pkt_fwd
9.	Packet Length Mean

Table 4-6 features return by fourth filtering using information gain

#	Feature
1.	Destination Port
2.	Protocol
3.	Total Length of Fwd Packets
4.	Min Packet Length
5.	Packet Length Std
6.	Source Port
7.	Init_Win_bytes_forward
8.	Packet Length Mean

Table 4-7 features return by fifth filtering using information gain

#	Feature
1.	Destination Port
2.	Protocol
3.	Total Length of Fwd Packets
4.	Min Packet Length
5.	Source Port
6.	Init_Win_bytes_forward
7.	Packet Length Mean

Table 4-8 features return by sixth filtering using information gain

#	Feature
1.	Protocol
2.	Total Length of Fwd Packets
3.	Min Packet Length
4.	Source Port
5.	Init_Win_bytes_forward
6.	Packet Length Mean

Table 4-9 features return by seventh filtering using information gain

#	Feature
1.	Total Length of Fwd Packets
2.	Min Packet Length
3.	Source Port
4.	Init_Win_bytes_forward
5.	Packet Length Mean

Table 4-10 features return by eightieth filtering using information gain

#	Feature
1.	Total Length of Fwd Packets
2.	Min Packet Length
3.	Source Port
4.	Packet Length Mean

4.2.3 Selection of Classifiers Algorithms and Their Performance

Selecting the appropriate ML algorithm for a specific task is a critical decision that can significantly impact the performance of the model. Choosing from a set of five different algorithms helps in finding the one that is best suited for a particular problem and dataset. In the current study, two phases were used to select the better algorithm to employ inside the architecture that has been defined in chapter three. The process of selecting the better algorithm has been done through two phases:

1. Phase one: Choosing the better two algorithms from five different algorithms depending on the accuracy and the time of training when classifying four different types of DDoS attacks which contain (Portmap, Syn, UDP, MSSQL) an addition to the bening packets.
2. Phase two: combine the selected algorithms through a stacked model to get better results.

4.2.3.1 Phase One

This section presents the results of the experimental model developed using the machine learning algorithms J48, MultilayerPerseptron, PART, Random Forest, and RepTree. As mentioned in section **(Performance Evaluation)** three metrics are used to measure the performance of the algorithm **(training time, Accuracy, and error rate)**. The experiment test modes split data into 60% training, 40% testing at the first test, and the second test, 70% training, 30% testing implemented. feature set 1 Table (4-2) is used in this phase and the first experiment (60,40) gave us better results in detecting normal packets.

The results of the first test are presented and discussed regarding evaluation metrics:

1- **Training time:** Table (4-11) illustrates that the highest training time belongs to MultilayerPerseptron (MP) and random forest (RF) algorithms respectively when using set 1 (Figure 1) which contains 12 features. In contrast, the lowest time registered to REPTree and J48 respectively when using the same set of features.

Table 4-11 illustrates the training time for 5 algorithms.

Algorithm	Training Time/s
J48	4.27
MP	220.72
REPTree	2.92
PART	7.38
RF	67.2

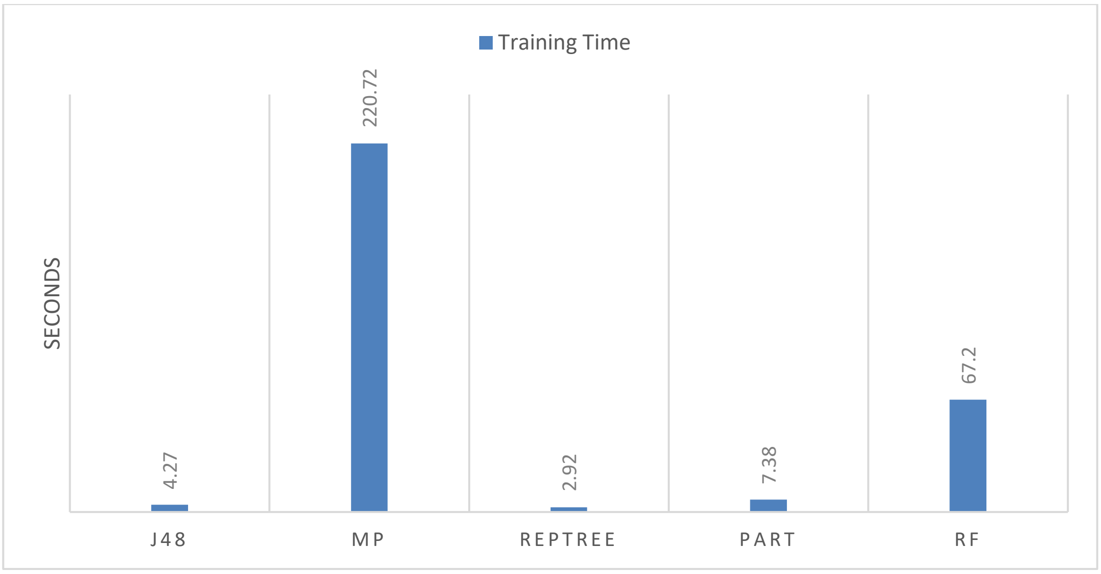


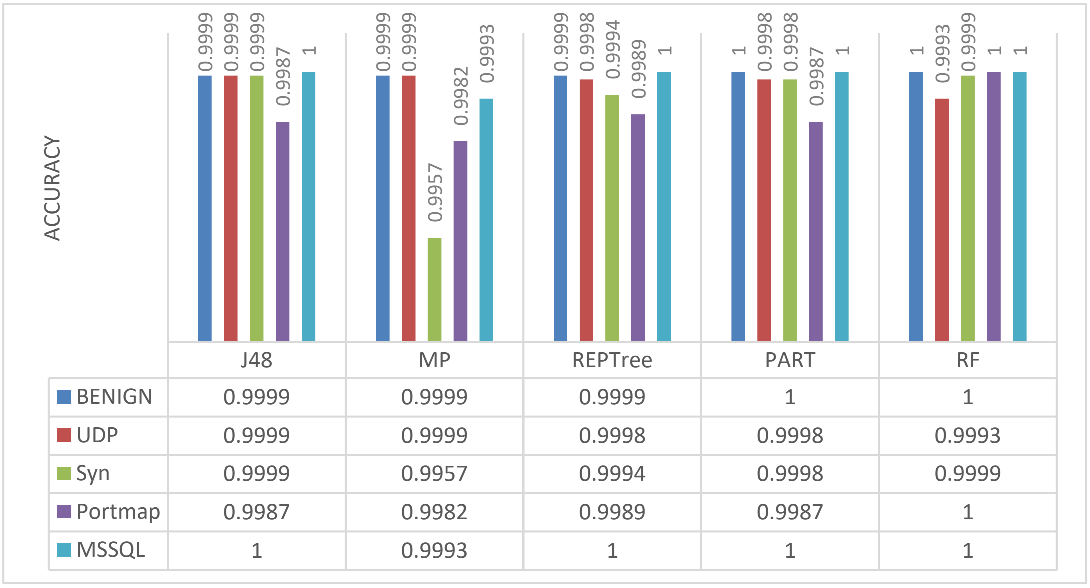
Figure 4-1 shows the differences between classifiers in training time

Training time is an important consideration in machine learning. It refers to the time taken to train a model on a given dataset. Faster training times can be beneficial for quick experimentation and deployment, while longer training times might be acceptable for more complex models if they yield better performance. From Figure 4.1, it is obvious to see a significant variation in training times across the different algorithms.

2- **Accuracy**: The analysis of the data reveals that five classifiers, J48, MultilayerPerseptron, RepTree, PART, and Random Forest have an overall accuracy greater than 99% when using features in set 1. RF gets the highest accuracy with 99.98% as presented in Table (4-12). Figures 4-2 to 4-7 present the confusion matrix for each classifier after implementation.

Table 4-12 Accuracy of the classifiers after implementation using features in Set 1

	Portmap	Syn	UDP	MSSQL	BENIGN
J48	**0.9999**	**0.9999**	**0.9999**	**0.9987**	**1.0000**
MP	**0.9999**	**0.9999**	**0.9957**	**0.9982**	**0.9993**
REPTree	**0.9999**	**0.9998**	**0.9994**	**0.9989**	**1.0000**
PART	**1.0000**	**0.9998**	**0.9998**	**0.9987**	**1.0000**
RF	**1.0000**	**0.9993**	**0.9999**	**1.0000**	**1.0000**



	J48	MP	REPTree	PART	RF
🟦 BENIGN	0.9999	0.9999	0.9999	1	1
🟥 UDP	0.9999	0.9999	0.9998	0.9998	0.9993
🟩 Syn	0.9999	0.9957	0.9994	0.9998	0.9999
🟪 Portmap	0.9987	0.9982	0.9989	0.9987	1
🟦 MSSQL	1	0.9993	1	1	1

Figure 4-2 Accuracy comparison between the used algorithms in detecting 4 types of attacks in addition to the benign packets

when diving deep through the data, the numbers show that all the classifiers successfully detected the BENIGN with 100% accuracy except MP. Portmap got the second-best detection result with an accuracy average of 99.99% by all the classifiers, in contrast, the MSSQL attack had the highest error rate in all the classifiers' results except with the RF classifier which classified it with 100% accuracy. All the other classifiers have acceptable results in most of the cases as illustrated in Table (4-12).

RF, PART, and J48 respectively had the best accuracy among all the classifiers With minor differences not exceeding 0.01 to 0.02 when using features of set 1. Table (4.13) to Table (4.15) illustrate the accuracy differences between the three mentioned classifiers using a confusion matrix for a better explanation.

Table 4-13 Confusion Matrix for RF algorithm

a	b	c	d	e	classified as
63998	0	0	0	2	a = Portmap
0	63987	2	10	1	b = Syn
0	0	63991	8	0	c = UDP
0	0	0	63996	0	d = MSSQL
0	0	0	0	62867	e = BENIGN

Table 4-14 Confusion Matrix for PART algorithm

a	b	c	d	e	classified as
63998	0	0	0	2	a = Portmap
0	63985	4	11	0	b = Syn
1	0	63990	6	2	c = UDP
0	0	82	63914	0	d = MSSQL
0	0	0	0	62867	e = BENIGN

Table 4-15 Confusion Matrix for J48 algorithm

a	b	c	d	e	classified as
63994	0	3	0	3	a = Portmap
0	63985	4	11	0	b = Syn
0	0	63991	8	0	c = UDP
0	0	82	63908	6	d = MSSQL
0	0	0	0	62867	e = BENIGN

3- **Error rate:** In machine learning, the error rate is a metric used to quantify the performance of a model by measuring the proportion of incorrect predictions it makes on a given dataset. It provides an overall view of how well the model is performing in terms of misclassifying instances. The error rate is calculated as:

Error Rate = (Number of Incorrect Predictions) / (Total Number of Predictions) -- 4.1

Alternatively, it can be expressed as:

Error Rate = 1 – Accuracy -- 4.2

The error rate of the used algorithm is represented in Table 4-16 and Figure 4-3:

Table 4-16 Overall error rate of each algorithm

algorithm	Error Rate
RF	0.000072
J48	0.000367
PART	0.000339
REPTree	0.000383
MP	0.001656

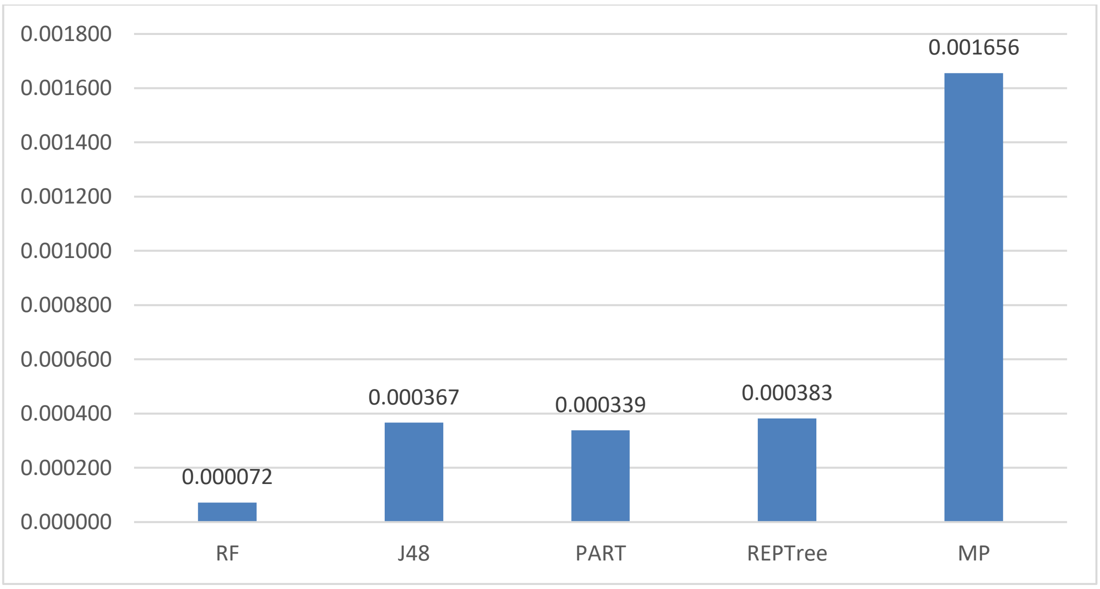


Figure 4-3 shows the comparison between algorithms based on the error rate

When building a stacked model, which involves combining predictions from multiple base models to create a more powerful ensemble, the choice of algorithms for the base models is crucial. The goal is to select diverse and complementary algorithms that have the potential to capture different aspects of the data and improve overall performance. In this study, RF and J48 were chosen because of the high accuracy of the RF and the low time of the J48.

In the next phase, only the two algorithms are used to build the targeted classification model.

4.2.3.2 Phase two

Building a stacked model to classify DDoS attacks involves several steps Figure 4-4.

1- **Selection of Base Models**: in this step, it is necessary to choose a diverse set of base models that can capture different patterns and characteristics of network traffic. Regarding the previous phase, the choice was **RF** and **J48**. Both algorithms are trained using datasets discussed in previous sections after dividing them into 60% for training and 40% for testing. Also in this phase, **Nine** sets of features were used to find the better set of features.

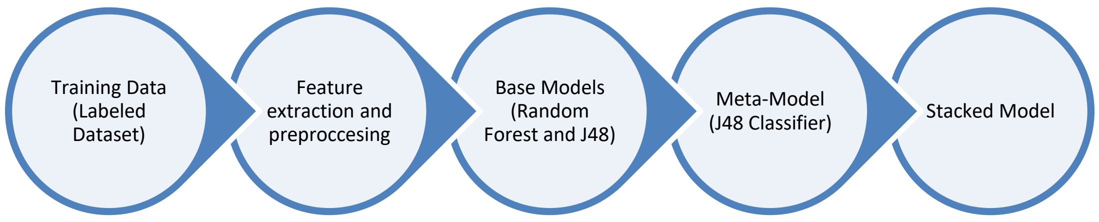

Training Data (Labeled Dataset)
Feature extraction and preproccesing
Base Models (Random Forest and J48)
Meta-Model (J48 Classifier)
Stacked Model

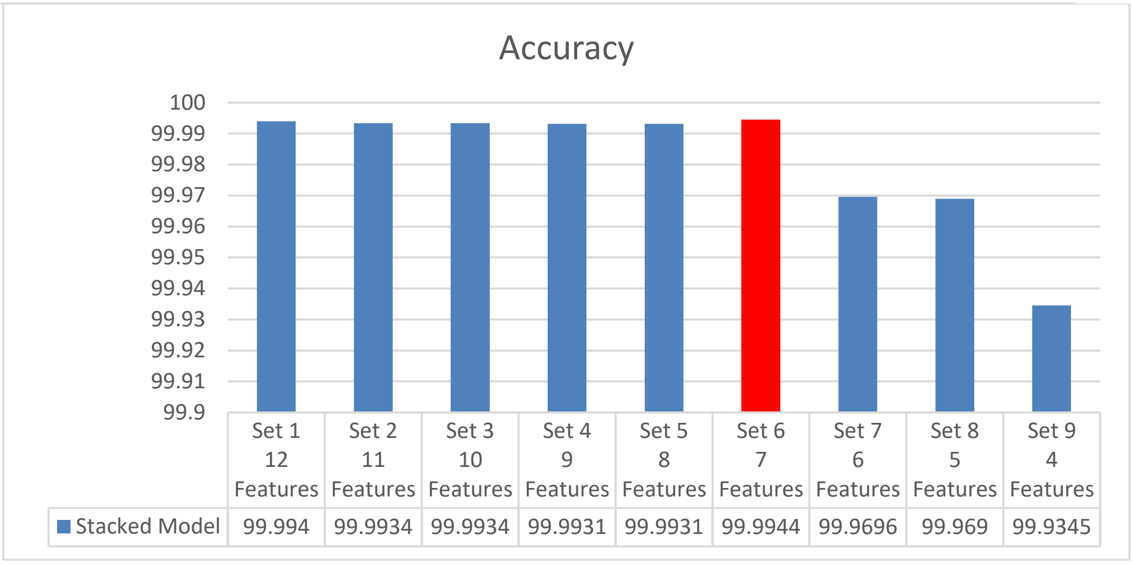
Figure 4-4 steps of creating a stacked model

2- **Generate Predictions from Base Models**: Using the trained base models, predictions were generated on the validation data. These predictions serve as input features for the meta-model.

3- **Building the Meta-Model**: for the meta-model, **J48** is chosen to combine the predictions from the base models. Then the meta-model is trained using the validation data, where the target variable is the true class label of the instances. When using a decision tree the tree is always pruned to optimize the rules.

4- **Evaluation and Testing**: Using the trained meta-model to make predictions on the testing dataset. The performance of the stacked model was evaluated using appropriate metrics such as **accuracy**, **precision**, **recall**, and **size of the tree** to choose the better set of features to give better results with the stacked model. The evaluation is described regarding the mentioned metrics as the following:

A. **Accuracy**: because of using 9 sets of features, the table contains 9 different value for accuracy, each value belong to one of the sets. By the accuracy, it is possible to define which set of features after implementing the stacked model is better. The values refer to the set that contains 7 features Figure 4-5.



	Set 1 12 Features	Set 2 11 Features	Set 3 10 Features	Set 4 9 Features	Set 5 8 Features	Set 6 7 Features	Set 7 6 Features	Set 8 5 Features	Set 9 4 Features
■ Stacked Model	99.994	99.9934	99.9934	99.9931	99.9931	99.9944	99.9696	99.969	99.9345

Figure 4-5 illustrates the accuracy of implementing the Stacked Model using 9 Sets

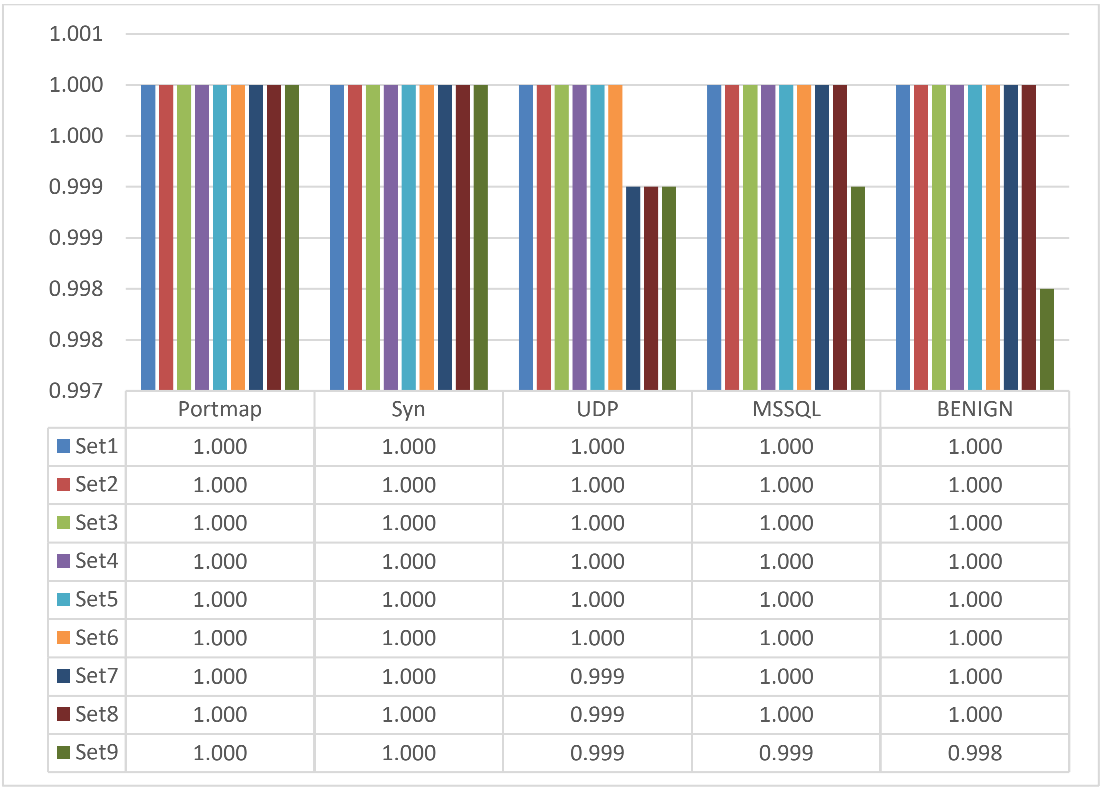
The confusion matrix of the stacked model when using set 6 (7 features) is illustrated in Table 4-17

Table 4-17 Illustrates the confusion matrix of the stacked model using set 6

a	b	c	d	e	classified as
63998	0	0	1	1	a = Portmap
0	63988	2	9	1	b = Syn
0	1	63995	3	0	c = UDP
0	0	0	63996	0	d = MSSQL
0	0	0	0	62867	e = BENIGN

B. **Precision**: An optimal classifier should ideally achieve a precision score of 1. This condition is met when the values of both the numerator and the denominator are identical, meaning True Positives (TP) equals TP + False Positives (FP). Consequently, a precision score of 1 implies that the value of False Positives (FP) is zero.

As illustrated in Figure (4-6), the majority of the precision values are (1) except in Sets 7,8, and 9. The Precision dropped to 0.999 in classifying UDP attacks by using the mentioned 3 sets. In set 9 the 0.9 precision is still gained when classifying the MSSQL attack and benign packets. Set 1 to Set 6 had precision with value (1) when classifying all the attack cases. The results indicate that when using precision as a performance metric three sets were excluded (set 7, set 8, and set 9).

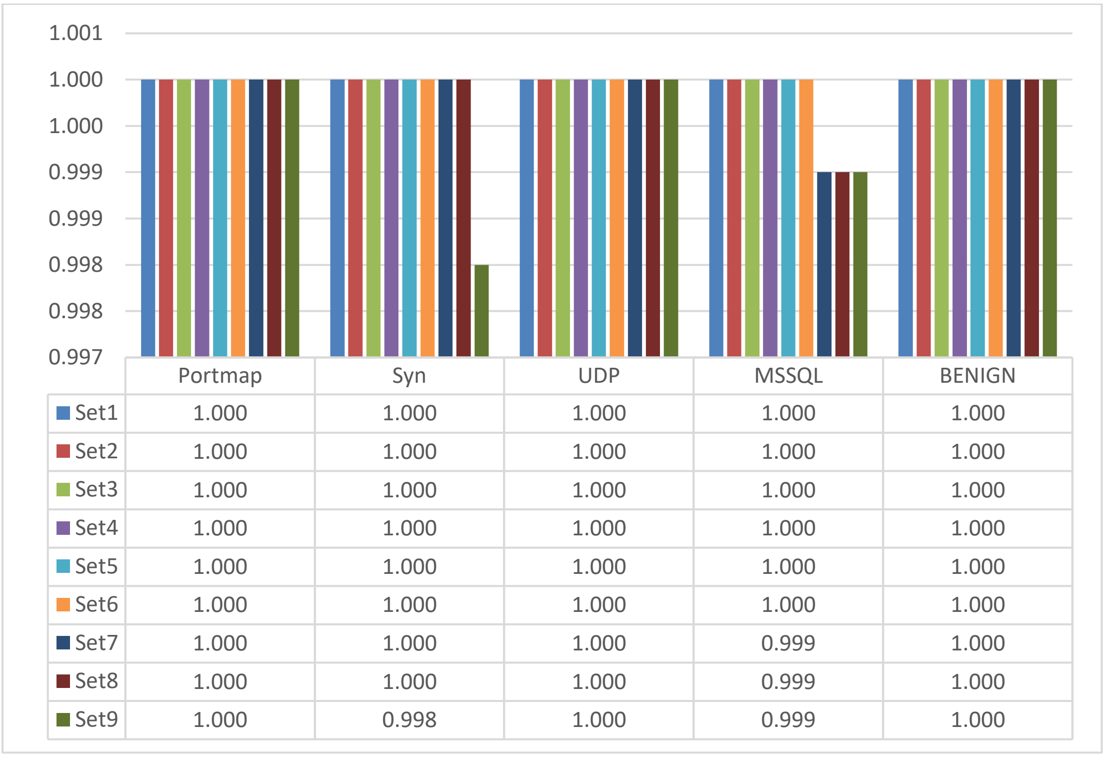


	Portmap	Syn	UDP	MSSQL	BENIGN
🟦 Set1	1.000	1.000	1.000	1.000	1.000
🟥 Set2	1.000	1.000	1.000	1.000	1.000
🟩 Set3	1.000	1.000	1.000	1.000	1.000
🟪 Set4	1.000	1.000	1.000	1.000	1.000
🟦 Set5	1.000	1.000	1.000	1.000	1.000
🟧 Set6	1.000	1.000	1.000	1.000	1.000
🟦 Set7	1.000	1.000	0.999	1.000	1.000
🟥 Set8	1.000	1.000	0.999	1.000	1.000
🟩 Set9	1.000	1.000	0.999	0.999	0.998

Figure 4-6 Precision of stacked model based on attack type using 9 sets of features.

C. **Recall**: The recall is calculated by dividing the number of Positive samples correctly classified as Positive by the total number of Positive samples. The recall reflects the classifier's ability to recognize positive samples. More affirmative samples are detected as recall increases.

In using all the sets, the classification results show that also in recall the features sets from 1 to 6 were better than the last three sets. See Figure 4-7.



	Portmap	Syn	UDP	MSSQL	BENIGN
Set1	1.000	1.000	1.000	1.000	1.000
Set2	1.000	1.000	1.000	1.000	1.000
Set3	1.000	1.000	1.000	1.000	1.000
Set4	1.000	1.000	1.000	1.000	1.000
Set5	1.000	1.000	1.000	1.000	1.000
Set6	1.000	1.000	1.000	1.000	1.000
Set7	1.000	1.000	1.000	0.999	1.000
Set8	1.000	1.000	1.000	0.999	1.000
Set9	1.000	0.998	1.000	0.999	1.000

Figure 4-73 shows the recall of the classifier based on attack type using 9 sets of features.

D. Size of Tree: The depth of a decision tree, also known as its "tree depth" or "max depth," is a hyperparameter that plays a crucial role in determining the performance and behavior of the decision tree model in machine learning. It refers to the maximum number of levels or nodes from the root node to the deepest leaf node in the tree. Pruning is the process of removing certain branches from a decision tree to improve its generalization performance and prevent overfitting. In all the experiments the meta-model was pruned to get less depth of tree as represented in Table 4-18.

Table 4-18 shows the size of the tree before and after pruning

Sets	**Size of the tree before pruning**	**Size of the tree after pruning**
Set 1	119	23
Set 2	119	13
Set 3	121	17
Set 4	121	17
Set 5	121	13
Set 6	129	17
Set 7	153	9
Set 8	153	9
Set 9	239	15

After examining the outcomes of four performance metrics, it is established that Set 6 achieves the most superior overall performance among the Sets. The Stacked model classifier achieved the highest accuracy of 99.9944% in comparison with other classifiers implemented in phase one. The rest of the classifiers are not bad but the stacked model had the highest accuracy score in identifying 4 types of DDoS Attacks (UDP, SYN, Portmap, and MSSQL).

To determine the better classifier, the best feature set should be determined first, according to the accuracy obtained.

In comparing the results achieved by this study with the results of other studies, it is clear that the stacked model of this study is better. See Table 4-19.

Table 4-19 comparisons between this study and existing systems.

#	Authors	Year of the study	Detection Accuracy	Features
1	[9]	2020	99.0	unknown
2	[11]	2020	98	Unknown
3	[12]	2020	99.7	Unknown
4	[88]	2020	99.79	22
5	[89]	2020	99.55%	10
6	[16]	2021	99.84	Unknown
7	[90]	2021	99.79%	15
			96.47%	4
8	[91]	2021	99.50%	unknown
9	[92]	2022	99.51	4
			99.96	8
10	[17]	2022	79.2 – 96.69	unknown
11	Proposed work	2023	99.9944	7

Regarding the results of the mentioned metrics, the stacked model was used as a classifier in the real-time proposed system.

4.3 The Proposed System in the NGN Environment

The following subsections present the proposed system implementation using NGN technology:

4.3.1 Proposed Topology and Physical Implementation

Network slicing is a technique that allows a single physical network infrastructure to be divided into multiple virtual network slices, each with its own isolated and customizable network resources. This concept is particularly relevant in 5G and edge computing scenarios where different services and applications require distinct network characteristics and resources.

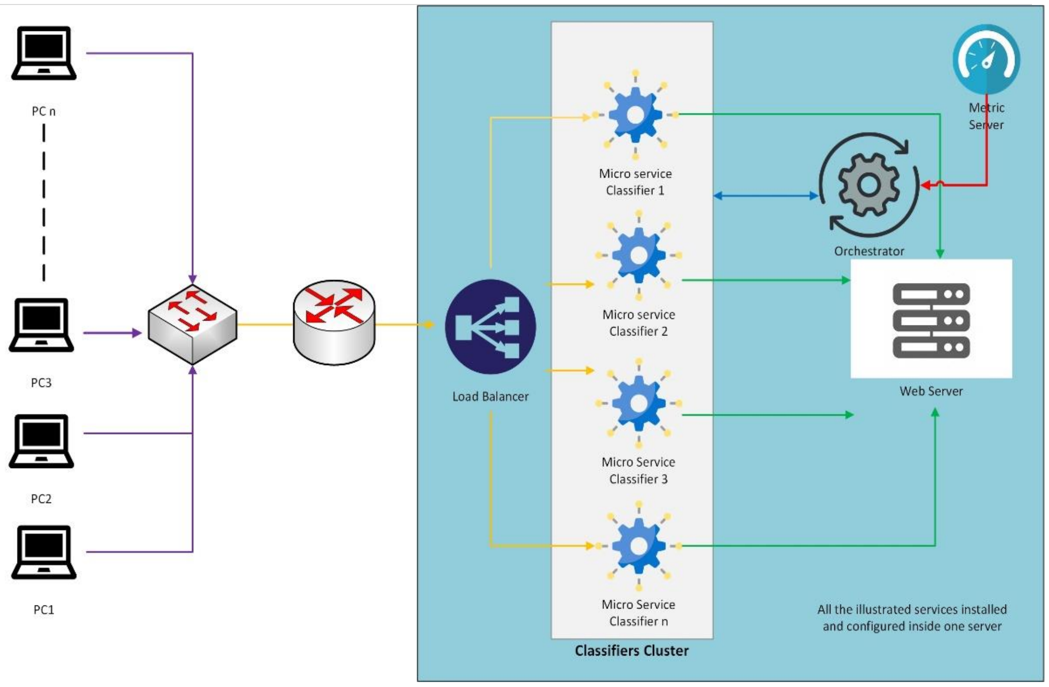

PC n
PC3
PC2
PC1
Load Balancer
Micro service Classifier 1
Micro service Classifier 2
Micro Service Classifier 3
Micro Service Classifier n
Classifiers Cluster
Orchestrator
Metric Server
Web Server
All the illustrated services installed and configured inside one server

Figure 4-84 Main diagram of the proposed system, illustrates all the components used in the system

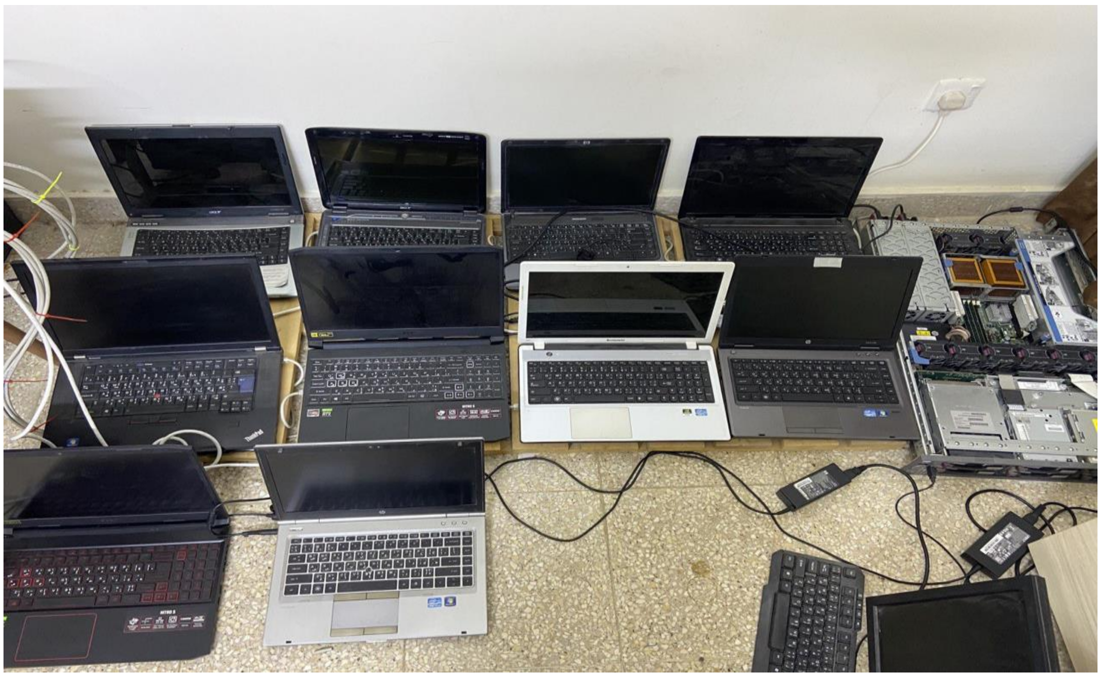


Figure 4-9 Physical implementation of the topology

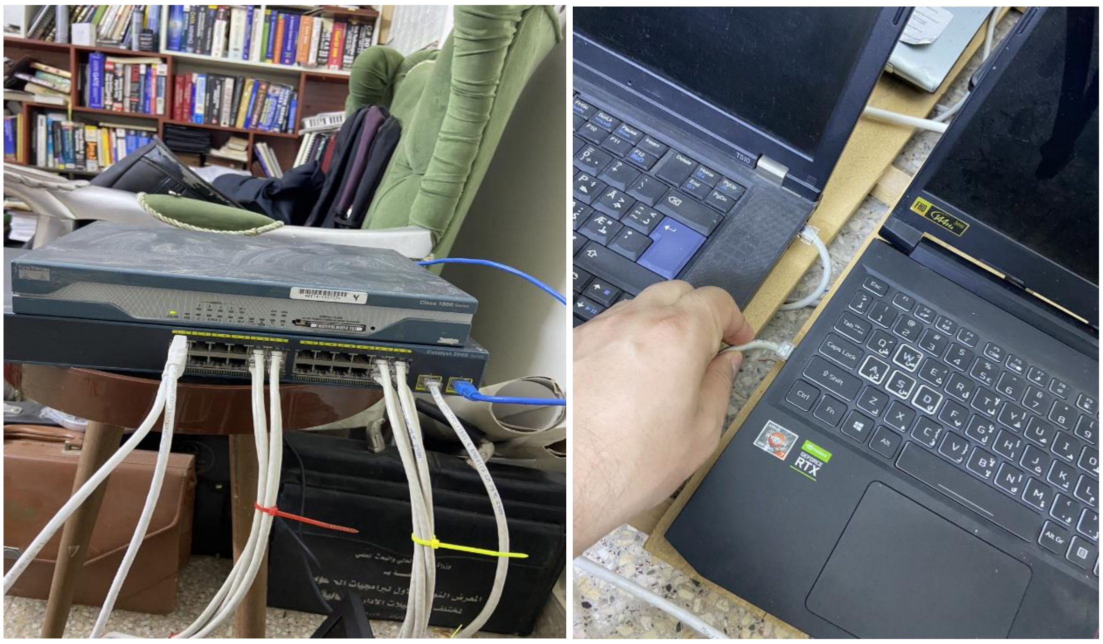


Figure 4-105 Cabeling connection between PCs and entermediray devices

To implement network slicing, Kubernetes, Calico, container orchestration, and networking tools are required to create and manage these isolated network slices. As Figures 4-8, 4-9, and 4-10 illustrate, the following tools have been used to reach the final goal:

1 DDoS attack source

2 Legitimate packet source

3 Router

4 Switch

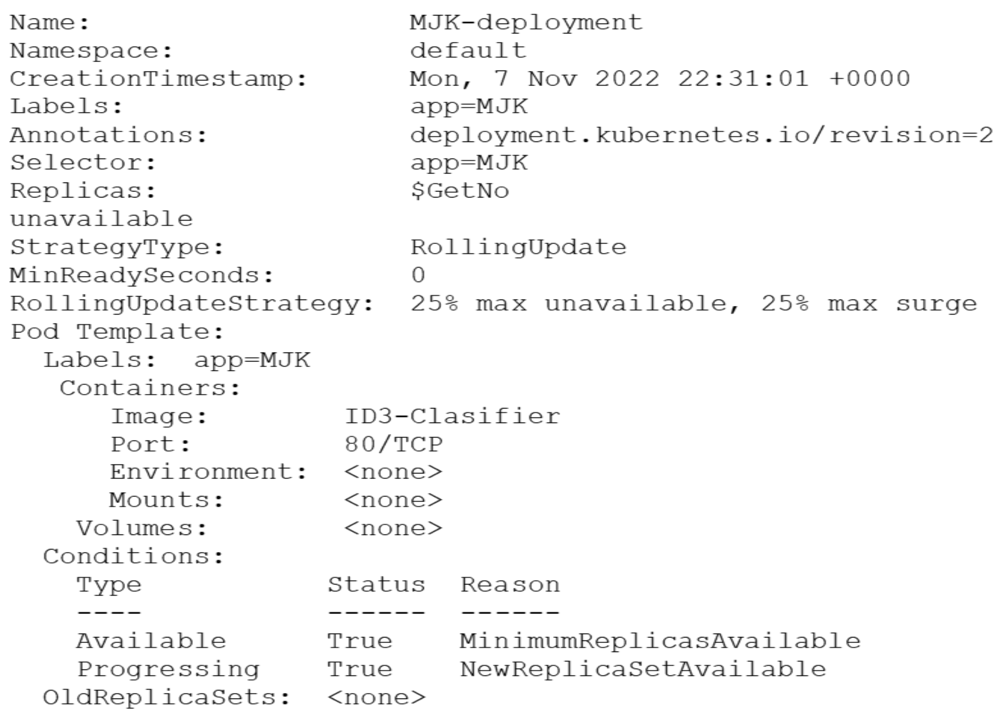
5 Server (Load Balancer, Kubernetes, Calico, Docker, Web server and Prometheus are installed inside the server)

4.3.2 Covered Attack Types

Machine learning-driven detection techniques within Next-Generation Networks (NGNs) have been developed to identify a range of attack categories. Kaspersky's reports [93]–[96] highlight that UDP flooding has overwhelmingly been the predominant attack type spanning from 2019 to the first quarter of 2022. Consequently, our approach is tailored to identifying both UDP and SYN DDoS attacks. Notably, our methodology possesses the capability to accommodate various flooding attack scenarios. Looking ahead, our research endeavors will extend to creating an extensive attack detection module capable of effectively addressing diverse attack types on a large scale.

4.3.3 Implementation of ML in the proposed system

At first, the ML model was packaged with necessary dependencies, as a Docker container image and made it able to run as a standalone service inside a container. The image is invoked by a YAML file with the name ***it-uobabylon-ddosclassifier.yaml.*** The file built as Figure 4-11



```
Name:                   MJK-deployment
Namespace:              default
CreationTimestamp:      Mon, 7 Nov 2022 22:31:01 +0000
Labels:                 app=MJK
Annotations:            deployment.kubernetes.io/revision=2
Selector:               app=MJK
Replicas:               $GetNo
unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=MJK
   Containers:
      Image:        ID3-Clasifier
      Port:         80/TCP
      Environment:  <none>
      Mounts:       <none>
    Volumes:        <none>
  Conditions:
    Type           Status  Reason
    ----           ------  ------
    Available      True    MinimumReplicasAvailable
    Progressing    True    NewReplicaSetAvailable
  OldReplicaSets:  <none>
```

Figure 4-11 Initiation code of microprocessor template

4.3.4 Dynamic allocation and auto-scaling

various parameters and mechanisms are used to scale up and scale down pods to meet the desired application requirements and resource constraints [97], [98]. The primary way to do this is by adjusting the desired number of replicas for a particular deployment, replica set, or stateful set which can be summarized as the following:

- **Replica Count**: Specify the desired number of replicas for a particular workload using the replicas field in the deployment, replica set, or stateful set YAML definition. Increasing this count scales up the pods while decreasing it scales down.
- **Autoscaling**: automatically adjusts the number of replicas based on metrics defined by the researcher.
- **Cluster AutoScaler**: Cluster AutoScaler is a feature that adjusts the number of nodes in the cluster based on resource demand. This indirectly affects pod scaling since more nodes allow for more pods to run.
- **Node Affinity and Anti-Affinity**: You can use node affinity and anti-affinity rules to influence the placement of pods on specific nodes or prevent them from running on the same node. This can impact scaling strategies by distributing pods across nodes more effectively.
- **Pod Priority and Preemption**: By setting pod priorities, the control of which pods are evicted when resource constraints are reached. Higher-priority pods may not be evicted, while lower-priority pods may be preempted to free up resources for higher-priority pods.

- **Taints and Tolerations**: Taints can be applied to nodes to repel certain pods, while tolerations allow pods to tolerate those taints. This can be used to influence pod placement and scaling decisions.
- **Pod Disruption Budgets**: It defined PodDisruptionBudgets to limit the number of pods that can be disrupted during scaling or maintenance operations, ensuring that a minimum number of pods are always available.
- **Manual Scaling**: manually scale the deployments using the CLI commands.

In this proposed work, auto-scaling is used by implementing dynamic allocation to adjust the number of replicas. The orchestrator replicates the pods depending on the incoming instruction from the metric server which calculates the continuing CPU and RAM consumption and sends feedback to the orchestrator to make a decision. In studies [40]-[42] the RAM or Number of waiting requests in the buffer or time series and ML techniques a metrics for auto-scalling. In experiments, the Combination of CPU and RAM consumption was more effective, regarding results in this proposed work.

The initial values for the parameters used by the metric server for dynamic allocation in the experiment are illustrated in Table 4- 20.

Table 4-20 shows the initial, minimum, and maximum values of the parameters used by the metric server for dynamic allocation

Parameter	Value	Notes
CPU cores	32 core	
RAM	24 GB	
The initial number of pods	1	The number of Pods increases depending on the evaluation of the orchestrator every 30 seconds using the metric server
Minimum number of pods	1	Each pod uses 1 Core
Maximum number of pods	28	
Minimum number of Cores for each pod	1000 mC	1000 mC = 1 Core
Maximum number of Cores for each pod	2000 mC	2000 mC = 2 Core
Maximum amount of RAM for each Pod	500 Mbit	

4.2.4 Implementing the proposed work

As presented in Figure 4-12, the number of Pods started as one Pod (initialized value) and increased regarding the incoming amount of attacks to absorb it. Also, the Figure shows that the attack (which lasted approximately two hours) was not at the same rate to test the flexibility of the system and respond by increasing or decreasing the number of Pods

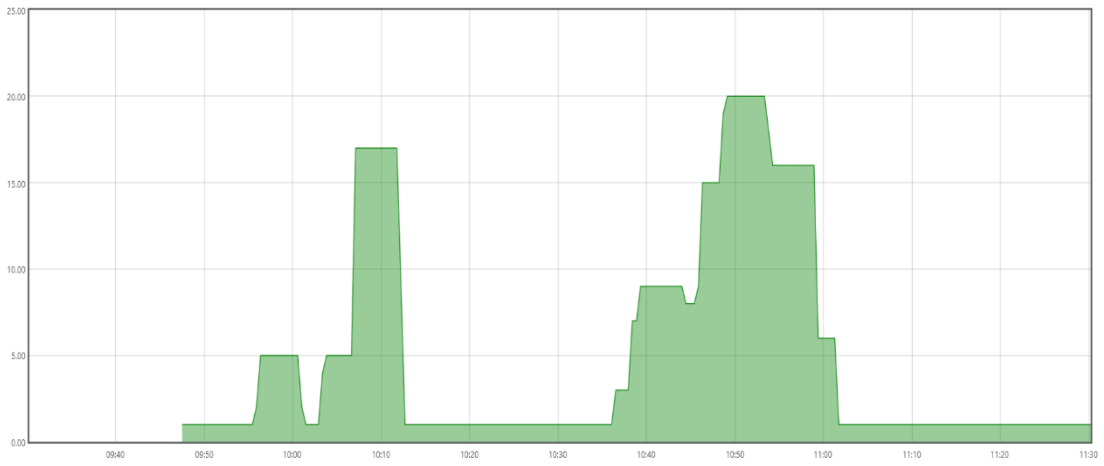


Figure 4-62 Increasing and decreasing of pods regarding the amount of attack

At the same time, the consumption of the resources was monitored too. In Figures 4-13 and 4-14, the resources Consumption by each Pod is colored by different colors to show that each increase of Pods, done by an increase in resource consumption

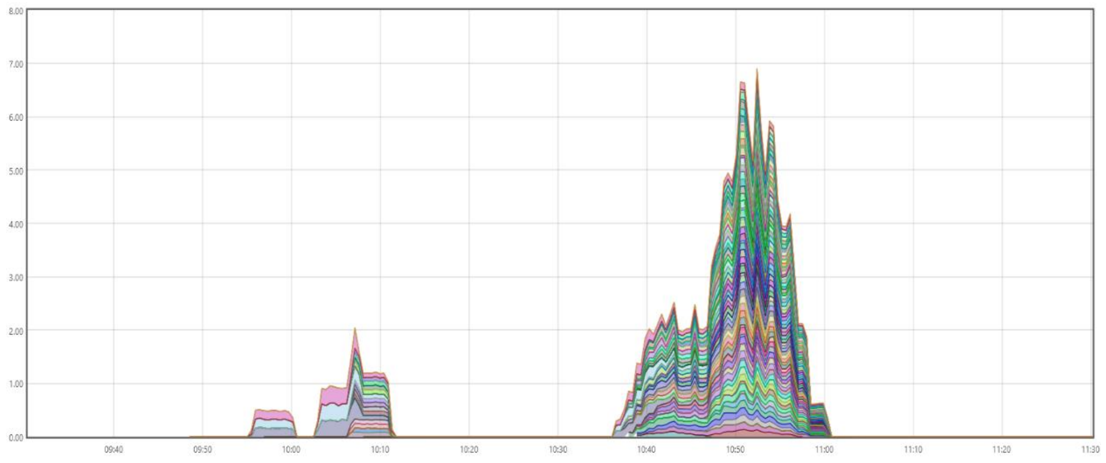


Figure 4-73 shows the CPU Consumed by Pods

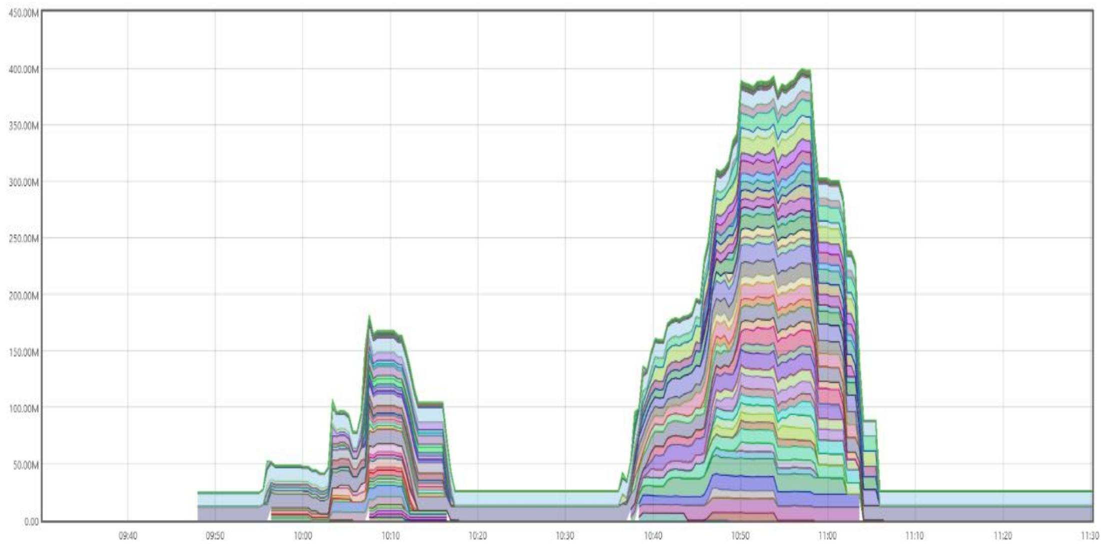
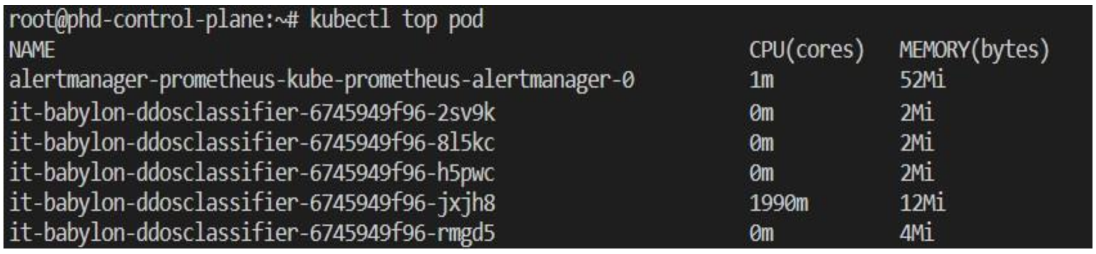


Figure 4-084 shows the RAM Consumed by Pods

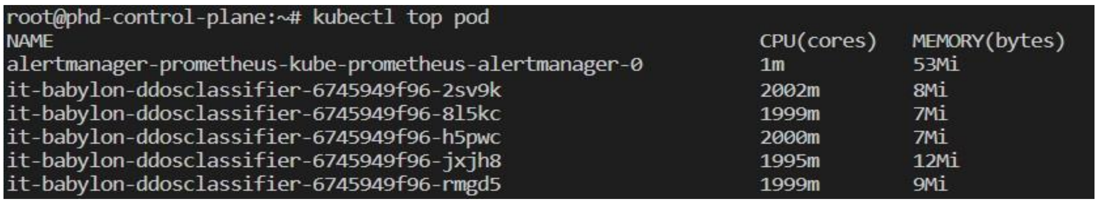
In the context of Kubernetes, replica refers to the number of identical pods that are managed by certain Kubernetes resources like Deployments, ReplicaSets, and StatefulSets. These resources ensure that a specified number of replicas (pod instances) are running at all times. By maintaining multiple replicas of the application, Kubernetes ensures that the application remains available even if some pods fail or become unavailable due to node issues or other factors. Figure 4-15 shows that Pods replicas (which was 1 as mentioned in Table 4-12) increased after the feedback of the metric server to the orchestrator. It is clear that all the replicas have the name of the image with different extensions to assign a unique ID for each Pod



```
root@phd-control-plane:~# kubectl top pod
NAME                                                CPU(cores)   MEMORY(bytes)
alertmanager-prometheus-kube-prometheus-alertmanager-0   1m      52Mi
it-babylon-ddosclassifier-6745949f96-2sv9k          0m           2Mi
it-babylon-ddosclassifier-6745949f96-8l5kc          0m           2Mi
it-babylon-ddosclassifier-6745949f96-h5pwc          0m           2Mi
it-babylon-ddosclassifier-6745949f96-jxjh8          1990m        12Mi
it-babylon-ddosclassifier-6745949f96-rmgd5          0m           4Mi
```

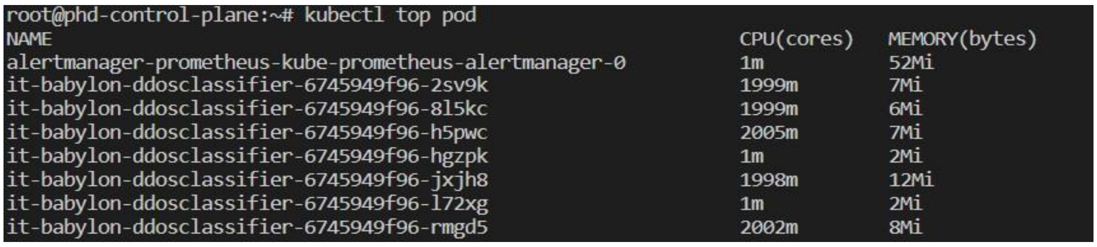
Figure 4-95 Scaling of the classifier from 1 to 5 to handle the attack

The scaling continues to increase until all the cores are consumed or the attack is handled Figures 4-16, 4-17, 4-18, 4-19



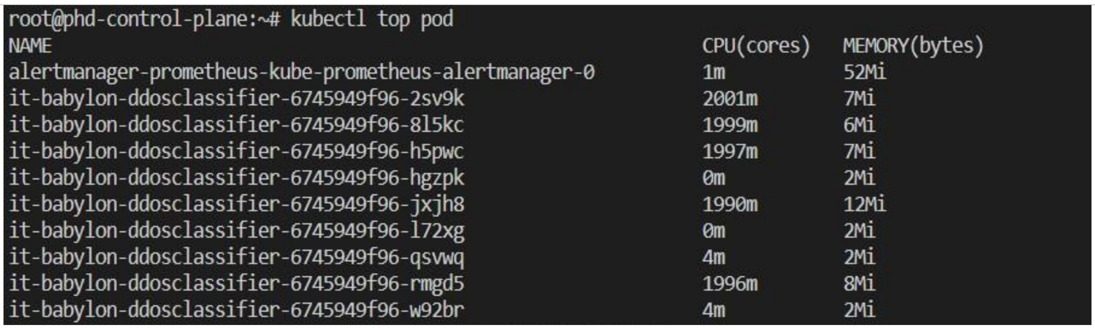
```
root@phd-control-plane:~# kubectl top pod
NAME                                                CPU(cores)   MEMORY(bytes)
alertmanager-prometheus-kube-prometheus-alertmanager-0   1m      53Mi
it-babylon-ddosclassifier-6745949f96-2sv9k          2002m        8Mi
it-babylon-ddosclassifier-6745949f96-8l5kc          1999m        7Mi
it-babylon-ddosclassifier-6745949f96-h5pwc          2000m        7Mi
it-babylon-ddosclassifier-6745949f96-jxjh8          1995m        12Mi
it-babylon-ddosclassifier-6745949f96-rmgd5          1999m        9Mi
```

Figure 4-1610 Scaling up of the classifier



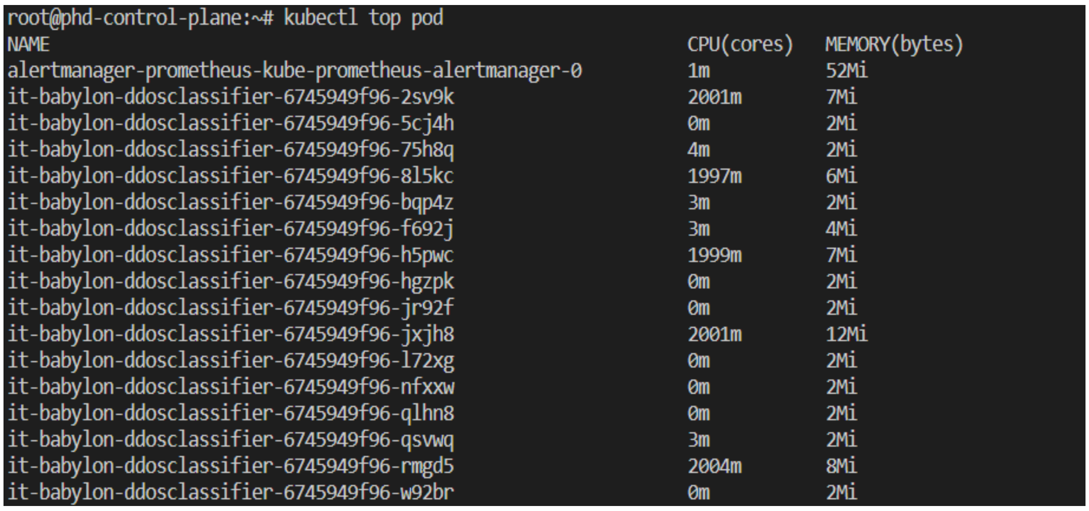
```
root@phd-control-plane:~# kubectl top pod
NAME                                                CPU(cores)   MEMORY(bytes)
alertmanager-prometheus-kube-prometheus-alertmanager-0   1m      52Mi
it-babylon-ddosclassifier-6745949f96-2sv9k          1999m        7Mi
it-babylon-ddosclassifier-6745949f96-8l5kc          1999m        6Mi
it-babylon-ddosclassifier-6745949f96-h5pwc          2005m        7Mi
it-babylon-ddosclassifier-6745949f96-hgzpk          1m           2Mi
it-babylon-ddosclassifier-6745949f96-jxjh8          1998m        12Mi
it-babylon-ddosclassifier-6745949f96-l72xg          1m           2Mi
it-babylon-ddosclassifier-6745949f96-rmgd5          2002m        8Mi
```

Figure 4-17 Scaling of the classifier when the attack increased



```
root@phd-control-plane:~# kubectl top pod
NAME                                                    CPU(cores)   MEMORY(bytes)
alertmanager-prometheus-kube-prometheus-alertmanager-0  1m           52Mi
it-babylon-ddosclassifier-6745949f96-2sv9k              2001m        7Mi
it-babylon-ddosclassifier-6745949f96-8l5kc              1999m        6Mi
it-babylon-ddosclassifier-6745949f96-h5pwc              1997m        7Mi
it-babylon-ddosclassifier-6745949f96-hgzpk              0m           2Mi
it-babylon-ddosclassifier-6745949f96-jxjh8              1990m        12Mi
it-babylon-ddosclassifier-6745949f96-l72xg              0m           2Mi
it-babylon-ddosclassifier-6745949f96-qsvwq              4m           2Mi
it-babylon-ddosclassifier-6745949f96-rmgd5              1996m        8Mi
it-babylon-ddosclassifier-6745949f96-w92br              4m           2Mi
```

Figure 4-18 Scaling of the classifier keeps increasing to 9 replicas

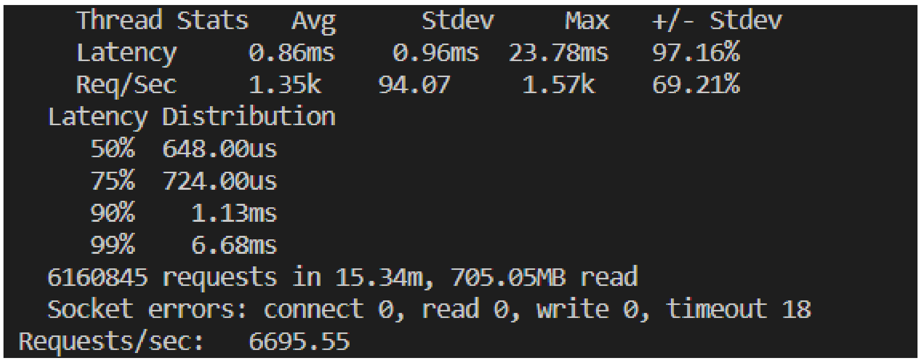


```
root@phd-control-plane:~# kubectl top pod
NAME                                                    CPU(cores)   MEMORY(bytes)
alertmanager-prometheus-kube-prometheus-alertmanager-0  1m           52Mi
it-babylon-ddosclassifier-6745949f96-2sv9k              2001m        7Mi
it-babylon-ddosclassifier-6745949f96-5cj4h              0m           2Mi
it-babylon-ddosclassifier-6745949f96-75h8q              4m           2Mi
it-babylon-ddosclassifier-6745949f96-8l5kc              1997m        6Mi
it-babylon-ddosclassifier-6745949f96-bqp4z              3m           2Mi
it-babylon-ddosclassifier-6745949f96-f692j              3m           4Mi
it-babylon-ddosclassifier-6745949f96-h5pwc              1999m        7Mi
it-babylon-ddosclassifier-6745949f96-hgzpk              0m           2Mi
it-babylon-ddosclassifier-6745949f96-jr92f              0m           2Mi
it-babylon-ddosclassifier-6745949f96-jxjh8              2001m        12Mi
it-babylon-ddosclassifier-6745949f96-l72xg              0m           2Mi
it-babylon-ddosclassifier-6745949f96-nfxxw              0m           2Mi
it-babylon-ddosclassifier-6745949f96-qlhn8              0m           2Mi
it-babylon-ddosclassifier-6745949f96-qsvwq              3m           2Mi
it-babylon-ddosclassifier-6745949f96-rmgd5              2004m        8Mi
it-babylon-ddosclassifier-6745949f96-w92br              0m           2Mi
```

Figure 4-19 Scaling of the classifier keeps increasing to 16 replicas

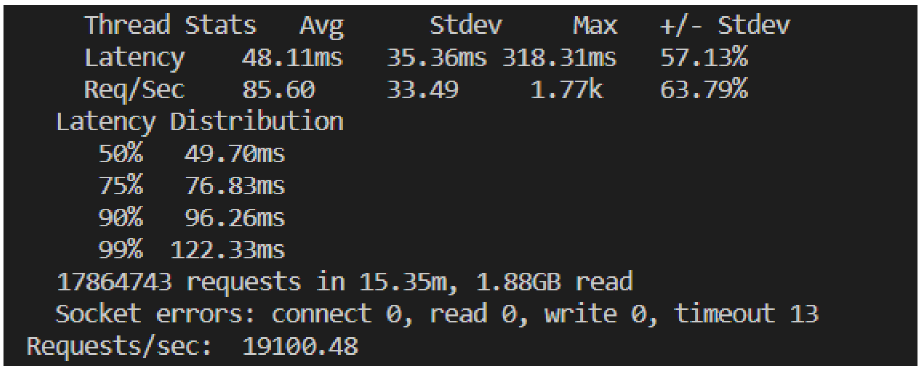
4.2.5 Metrics

To demonstrate the impact of the attack on the network, the study assessed changes in CPU utilization and data transfer rates on the control plane before and after the attack. The selected metrics for gauging the attack's effectiveness encompassed CPU usage, link latency, RAM utilization, and network bandwidth consumption. Furthermore, the study employed Prometheus to measure CPU consumption. Link latency refers to the time taken by a packet to traverse a link; the research measured link latency between an arbitrary host and the web server under two conditions: pre-attack and during the attack, as illustrated in Figures (4.20) and (4.21).



```
    Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     0.86ms    0.96ms  23.78ms   97.16%
    Req/Sec     1.35k    94.07     1.57k    69.21%
  Latency Distribution
     50%  648.00us
     75%  724.00us
     90%    1.13ms
     99%    6.68ms
  6160845 requests in 15.34m, 705.05MB read
  Socket errors: connect 0, read 0, write 0, timeout 18
Requests/sec:   6695.55
```

Figure 4-20 Latency before the attack



```
    Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    48.11ms   35.36ms 318.31ms   57.13%
    Req/Sec    85.60     33.49     1.77k    63.79%
  Latency Distribution
     50%   49.70ms
     75%   76.83ms
     90%   96.26ms
     99%  122.33ms
  17864743 requests in 15.35m, 1.88GB read
  Socket errors: connect 0, read 0, write 0, timeout 13
Requests/sec:  19100.48
```

Figure 4-21 Latency during the attack

For CPU consumption, as mentioned Promethuos is used, and the Consumption is measured when one classifier is used and multi classifier is used Figure 4.22

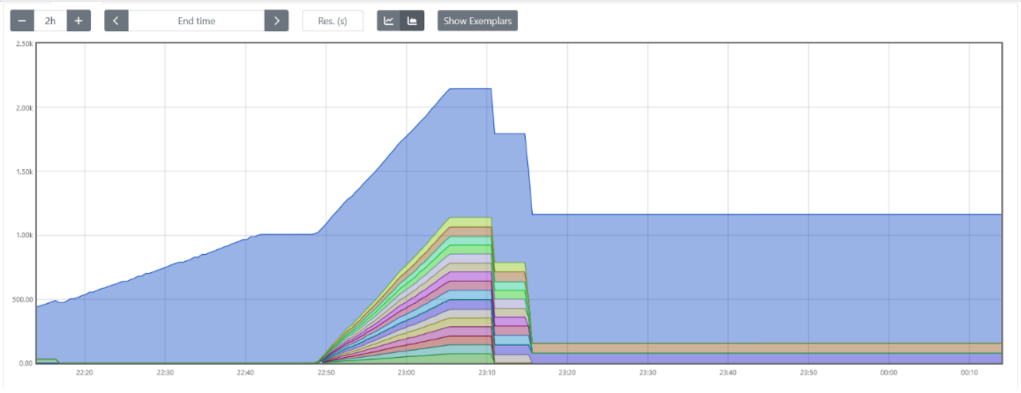


Figure 4-22 Comparison between CPU consumption when a single classifier (blue area) is used and when a multi-classifier is used

To measure memory consumption the same previous tool is used, and the histogram shows that the multi-classifier is better than single single-classifier Figure 4-23

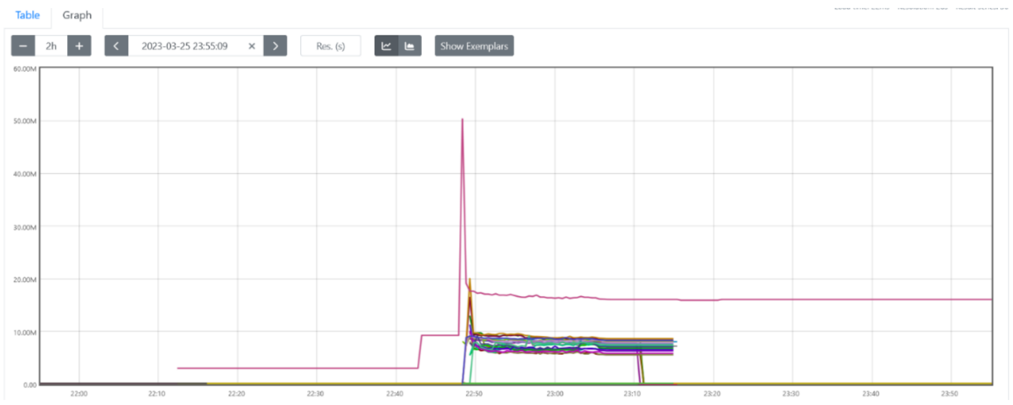


Figure 4-23 shows that a single classifier (the Pink line) consumes more memory than multi-classifiers (the different colors lines)

For network bandwidth, also the same tools are used to get the read of two stats, the first one when a single pod is implemented and the second one when a multi-pod is implemented Figure 4-24

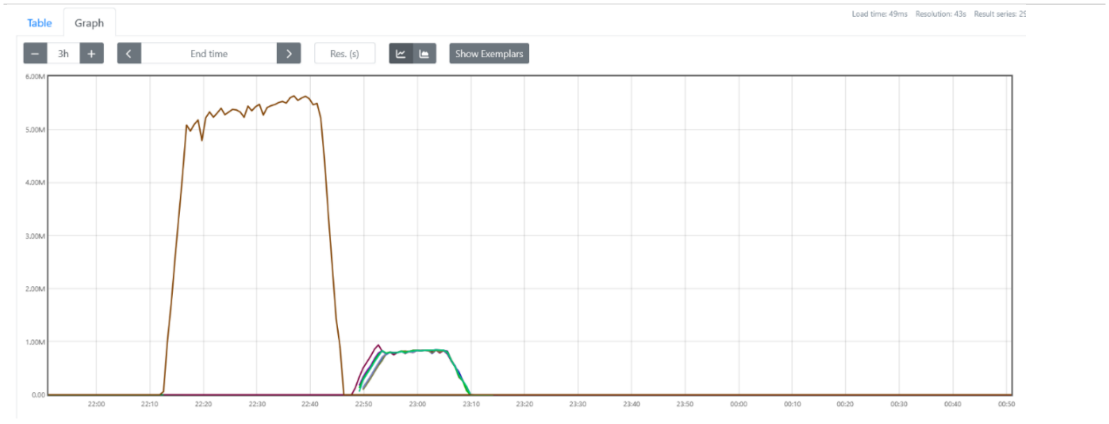


Figure 4-24 Comparison between single-pod bandwidth consumption and multi-pod bandwidth consumption

4.2.6 Results Discussion of the DDoS Effect

Figures (4-12 to 4-24) illustrate the performance metrics during the occurrence of a DDoS attack. When the attack is in progress, a noticeable upward trend in the average metric values becomes evident as the number of attacking hosts in the DDoS attack increases.

Conversely, the latency of links and flow rule instances was gauged through replay monitoring. Upon analyzing the CPU usage, it becomes evident that the multi-classifier's resource consumption outperforms that of the single-classifier. This observation holds true for RAM and bandwidth consumption as well. Notably, the dynamic allocation mechanism demonstrates remarkable efficiency in managing the ongoing attack.

4.3 Accuracy of the Proposed Work

To calculate the accuracy of the proposed system, the normal packets are calculated on the server side and compared with the packets sent by the client PCs in the same network. The final results of this study can be summarized in Table 4-21

Table 4-21 Results of implementing the proposed system

Normal traffic Packet/second	DDoS traffic packet/second	Total microservices	The success rate of normal packets
1000	0	1	99.3%
1000	1000	1	90.4%
1000	2000	1	80.9%
1000	8000	1	53.2%
1000	100000	1	7.06%
1000	1000	1-4	97%
1000	2000	4-6	97.1%
1000	8000	4-12	95.6%
1000	100000	12-20	95.8% - 95.1%

The thing that gives this work an advantage over the other studies mentioned in [26] and [31]-[39] is that all the other studies didn't use ML in their works and all of them used a VM as deployment technology which needs more resources than the microservices. Furthermore, the technology that has been used to create a slice in the mentioned works and the works [45]-[48] is more expensive and needs more time for configuration.

4.4 Advantages and Drawbacks of Virtual Networks against Physical Networks

Through the experiments, there were advantages for virtual networks but also there were drawbacks. In Table 4-18, a comparison has been made between virtual networks and physical networks as one of the results that have been recorded in this study.

Table 4-22 Comparision between virtual networks and physical networks

Subject	Virtual Network	Physical Network
Resource Efficiency	Virtual networks can share physical infrastructure resources, leading to better resource utilization. Multiple virtual networks can coexist on the same physical hardware, optimizing resource allocation	Physical networks have dedicated hardware resources for each network, which might lead to underutilization when resources are not fully consumed
Cost Savings	Virtual networks often lead to cost savings as they allow for more efficient use of hardware resources. This is particularly beneficial in cloud environments where resources can be dynamically allocated	Physical networks might require more hardware resources, leading to higher initial and maintenance costs
Performance	Virtual networks can experience slightly lower performance due to the overhead of virtualization. However, advancements in virtualization technologies have minimized this gap.	Physical networks generally offer better raw performance since there is no virtualization layer.
Isolation	Virtual networks provide isolated environments for different applications or tenants, but there's still a potential for "noisy neighbor" issues if not properly managed.	Physical networks can offer better isolation since resources are not shared, but achieving isolation might require separate hardware.
Control	Virtual networks allow for more granular control and configuration due to software-defined networking (SDN) capabilities. Administrators can define and manage network policies more flexibly.	Physical networks might have limitations in terms of control and might require manual configuration for changes.
Scalability	Virtual networks offer better scalability due to the ease of creating and managing new instances. Scaling can be automated, making it efficient and dynamic.	Scaling physical networks might require more manual intervention, especially when adding new hardware.

Subject	Virtual Network	Physical Network
Resource Utilization	Virtual networks can optimize resource utilization by dynamically allocating resources based on demand. Underutilization of resources is less likely.	Physical networks might experience resource wastage when resources are not fully utilized.
Deployment Time	Virtual networks can be quickly provisioned and deployed, making them suitable for dynamic workloads and rapid application development.	Physical networks generally require more time for setup, provisioning, and deployment due to hardware configurations.

CHAPTER Five

Conclusions and Future Work

5.1 Conclusions

NGN is an advanced telecommunications network infrastructure that is designed to provide enhanced communication services, improved performance, flexibility, and scalability compared to traditional legacy networks.

In the proposed work, the target is to enhance the security in NGNs using ML and dynamic allocation. several algorithms are implemented in the offline mode to select the best two algorithms to combine them in a stacked model. The stacked model which used RF and J48 algorithms has an accuracy of 99.9944% in offline mode. The new model is containerized inside a docker image and deployed as a classifier to classify packets in real-time to detect and prevent DDoS attacks in next-generation networks. The combination of Kubernetes and machine learning (ML) techniques holds significant promise in preventing DDoS attacks and enhancing the resilience of digital systems. The proposed system provides dynamic resource allocation and scaling capabilities and the experiment proves that it is an efficient way to handle the attack, while ML classifiers analyze network traffic patterns and classify the DDoS attacks to pass the legitimate packets or drop the malicious packets.

Anomaly Detection, Adaptive Routing and Load Balancing, Behavioral Analysis, Resource Isolation, and Threat Intelligence Integration, gave the system the ability to work fast and accurately. The best accuracy reached by the proposed system is between 97.3% to 95.1% regarding the strength of the attack which was between 1000 to 100000 packets per second.

Due to the standalone design, the requirement for DevOps training, and the constraints in deploying Next Generation Networks, many organizations and facilities struggle to fully implement and evaluate security systems. More effective and dependable instances of security could emerge as more devices become interconnected and seamlessly integrated across a wide range of real-time services.

5.2 Future Works

Numerous additional security issues and complexities may emerge in the future. As part of future endeavors, there is a plan to further develop the suggested system component within the DDoS mitigation framework, aiming to create a more authentic environment. Moreover, there is potential for reimagining the proposed approach to attain one or more of the following objectives.

- Real-Time ML Model Updates: Investigate methods for seamlessly updating ML models in real time based on the dynamic nature of network traffic and attack behavior. This could involve implementing an automated pipeline that continuously retrains and deploys ML models within the Kubernetes infrastructure to adapt to emerging threats.
- Zero-Day DDoS Attack Detection: Research could focus on leveraging ML-driven zero-day attack detection to identify DDoS attacks that exploit unknown vulnerabilities. Developing models that can quickly learn and identify anomalous traffic associated with previously unseen attack vectors would be a valuable future direction.
- Integrated Threat Intelligence: Future work could involve integrating threat intelligence platforms directly with Kubernetes and ML systems. This integration would enable automated threat data ingestion, allowing ML models to learn from and adapt to the latest attack trends and signatures.
- Hybrid Defense Strategies: Investigate hybrid defense strategies that combine ML-driven detection and Kubernetes-based dynamic

resource allocation with traditional DDoS protection techniques, such as rate limiting, packet filtering, and IP reputation lists. This holistic approach could provide a multi-layered defense against a wide range of attack vectors.

- Multi-Modal Data Fusion: Combine multiple data sources, such as network flow data, system logs, and application-layer metrics, to create a comprehensive view of the network environment. Integrating diverse data modalities can enhance the accuracy of ML models and provide a more complete understanding of ongoing attacks.
- User-Centric Adaptive Responses: Future research could explore ML models that adaptively adjust responses based on user profiles and behavior. This could help minimize disruption to legitimate users during attack mitigation and ensure a more personalized defense approach.
- Collaborative Defense Mechanisms: Investigate collaborative defense mechanisms where multiple Kubernetes clusters or organizations share threat intelligence and coordinate responses to large-scale DDoS attacks. Such collaboration could improve overall network resilience.

REFERENCES

[1] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines, “5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges,” *Comput. Networks*, vol. 167, p. 106984, 2020, doi: 10.1016/j.comnet.2019.106984.

[2] M. Agiwal, A. Roy, and N. Saxena, “Next generation 5G wireless networks: A comprehensive survey,” *IEEE Commun. Surv. Tutorials*, vol. 18, no. 3, pp. 1617–1655, Jul. 2016, doi: 10.1109/COMST.2016.2532458.

[3] J. G. Andrews *et al.*, “What will 5G be?,” *IEEE J. Sel. Areas Commun.*, vol. 32, no. 6, pp. 1065–1082, 2014, doi: 10.1109/JSAC.2014.2328098.

[4] A. Osseiran *et al.*, “Scenarios for 5G mobile and wireless communications: The vision of the METIS project,” *IEEE Commun. Mag.*, vol. 52, no. 5, pp. 26–35, May 2014, doi: 10.1109/MCOM.2014.6815890.

[5] Cisco, “Cisco Annual Internet Report - Cisco Annual Internet Report (2018 - 2023) White Paper,” 2020. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html (accessed May 02, 2022).

[6] M. A. Ali and A. A. Barakabitze, “Evolution of LTE and Related Technologies towards IMT-Advanced,” *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 5, no. 1, 2015, Accessed: Apr. 02, 2022. [Online]. Available: www.ijarcsse.com.

[7] R. Wazirali, T. Abu-ain, and R. Ahmad, “Digital-care in next generation networks : Requirements and future directions,” vol. 224, no. October 2022, 2023, doi: 10.1016/j.comnet.2023.109599.

[8] H. Ullah, N. G. Nair, A. Moore, C. Nugent, P. Muschamp, and M. Cuevas, “5G Communication : An Overview of Vehicle-to-Everything , Drones , and Healthcare,” *IEEE Access*, vol. PP, no. c, p. 1, 2019, doi: 10.1109/ACCESS.2019.2905347.

[9] M. S. Elsayed, N. A. Le-Khac, S. Dev, and A. D. Jurcut, “DDoSNet: A Deep-Learning Model for Detecting Network Attacks,” *Proc. - 21st IEEE Int. Symp. a World Wireless, Mob. Multimed. Networks, WoWMoM 2020*, pp. 391–396, 2020, doi: 10.1109/WoWMoM49955.2020.00072.

[10] “IDS 2018 | Datasets | Research | Canadian Institute for Cybersecurity | UNB.” https://www.unb.ca/cic/datasets/ids-2018.html (accessed Jul. 10, 2023).

[11] I. Bolodurina, A. Shukhman, D. Parfenov, A. Zhigalov, and L. Zabrodina, “Investigation of the problem of classifying unbalanced datasets in identifying distributed denial of service attacks,” *J. Phys. Conf. Ser.*, vol. 1679, no. 4, 2020, doi: 10.1088/1742-6596/1679/4/042020.

[12] T. Aytaç, M. A. Aydın, and A. H. Zaim, “Detection DDOS attacks using machine learning methods,” *Electrica*, vol. 20, no. 2, pp. 159–167, 2020, doi: 10.5152/electrica.2020.20049.

[13] A. E. Cil, K. Yildiz, and A. Buldu, “Detection of DDoS attacks with feed forward based deep neural network model,” *Expert Syst. Appl.*, vol. 169, p. 114520, 2021, doi: https://doi.org/10.1016/j.eswa.2020.114520.

[14] G. S. Kushwah and V. Ranga, “Optimized extreme learning machine for detecting DDoS attacks in cloud computing,” *Comput. Secur.*, vol. 105, p. 102260, 2021, doi: 10.1016/j.cose.2021.102260.

[15] H. Kousar, M. M. Mulla, P. Shettar, and D. G. Narayan, “Detection of

DDoS Attacks in Software Defined Network using Decision Tree," in *2021 10th IEEE International Conference on Communication Systems and Network Technologies (CSNT)*, 2021, pp. 783–788, doi: 10.1109/CSNT51715.2021.9509634.

[16] W. G. Gadallah, N. M. Omar, and H. M. Ibrahim, "Machine Learning-based Distributed Denial of Service Attacks Detection Technique using New Features in Software-defined Networks," *Comput. Netw. Inf. Secur.*, vol. 3, pp. 15–27, 2021, doi: 10.5815/ijcnis.2021.03.02.

[17] A. M. Araujo, A. Bergamini De Neira, M. Nogueira, and A. Medeiros Araujo, "Autonomous machine learning for early bot detection in the internet of things Autonomous machine learning for early bot detection in the internet of things," 2022, doi: 10.1016/j.dcan.2022.05.011.

[18] "IDS 2017 | Datasets | Research | Canadian Institute for Cybersecurity | UNB." https://www.unb.ca/cic/datasets/ids-2017.html (accessed May 02, 2023).

[19] R. Silipo RosariaSilipo, knimecom Iris Adae, A. Hart, and M. Berthold, "Seven Techniques for Dimensionality Reduction," 2014, Accessed: Jul. 11, 2023. [Online]. Available: http://www.sigkdd.org/kdd-cup-2009-customer-relationship-prediction.

[20] G. Lucky, F. Jjunju, and A. Marshall, "A Lightweight Decision-Tree Algorithm for detecting DDoS flooding attacks," doi: 10.1109/QRS-C51114.2020.00072.

[21] P. Maniriho, L. J. Mahoro, E. Niyigaba, Z. Bizimana, and T. Ahmad, "Detecting Intrusions in Computer Network Traffic with Machine Learning Approaches," *Int. J. Intell. Eng. Syst.*, vol. 13, no. 3, 2020, doi: 10.22266/ijies2020.0630.39.

[22] S. Peneti and Hemalatha, "DDOS Attack Identification using Machine

Learning Techniques,” *2021 Int. Conf. Comput. Commun. Informatics, ICCCI 2021*, Jan. 2021, doi: 10.1109/ICCCI50826.2021.9402441.

[23] M. H. Zaib, F. Bashir, K. N. Qureshi, S. Kausar, M. Rizwan, and G. Jeon, “Deep learning based cyber bullying early detection using distributed denial of service flow,” *Multimed. Syst.*, vol. 28, no. 6, pp. 1905–1924, Dec. 2022, doi: 10.1007/S00530-021-00771-Z.

[24] M. Ahsan, R. Gomes, M. M. Chowdhury, and K. E. Nygard, “Enhancing Machine Learning Prediction in Cybersecurity Using Dynamic Feature Selector,” *J. Cybersecurity Priv. 2021, Vol. 1, Pages 199-218*, vol. 1, no. 1, pp. 199–218, Mar. 2021, doi: 10.3390/JCP1010011.

[25] D. Kshirsagar and S. Kumar, “A feature reduction based reflected and exploited DDoS attacks detection system,” *J. Ambient Intell. Humaniz. Comput.*, vol. 13, no. 1, pp. 393–405, Jan. 2021, doi: 10.1007/S12652-021-02907-5.

[26] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, “Bohatei: Flexible and Elastic DDoS Defense Bohatei: Flexible and Elastic DDoS Defense,” *24th USENIX Secur. Symp.*, 2015, Accessed: Jun. 17, 2023. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/fayaz.

[27] “OpenDaylight.” https://www.opendaylight.org/ (accessed Jul. 04, 2023).

[28] “Open vSwitch.” https://www.openvswitch.org/ (accessed Jul. 04, 2023).

[29] M. Roesch, “Snort-Lightweight Intrusion Detection for Networks,” *Proc. LISA ’99 13 th Syst. Adm. Conf.*, 1998, [Online]. Available:

https://www.usenix.org/legacy/event/lisa99/full_papers/roesch/roesch.pdf.

[30] V. Paxson, “Bro: A System for Detecting Network Intruders in Real-Time,” *Proc. 7th USENIX Secur. Symp.*, 1998, [Online]. Available: https://www.usenix.org/legacy/publications/library/proceedings/sec98/full_papers/paxson/paxson.pdf.

[31] “netfilter/iptables project homepage - The netfilter.org ‘iptables’ project.” https://netfilter.org/projects/iptables/ (accessed Jul. 04, 2022).

[32] C. J. Fung and B. McCormick, “VGuard: A distributed denial of service attack mitigation method using network function virtualization,” *Proc. 11th Int. Conf. Netw. Serv. Manag. CNSM 2015*, pp. 64–70, Dec. 2015, doi: 10.1109/CNSM.2015.7367340.

[33] A. H. M. Jakaria, W. Yang, B. Rashidi, C. Fung, and M. A. Rahman, “VFence: A Defense against Distributed Denial of Service Attacks using Network Function Virtualization,” 2016, doi: 10.1109/COMPSAC.2016.219.

[34] B. Rashidi, C. Fung, and E. Bertino, “A Collaborative DDoS Defence Framework Using Network Function Virtualization,” *IEEE Trans. Inf. FORENSICS Secur.*, vol. 12, no. 10, 2017, doi: 10.1109/TIFS.2017.2708693.

[35] L. Zhou and H. Guo, “Applying NFV/SDN in mitigating DDoS attacks,” *IEEE Reg. 10 Annu. Int. Conf. Proceedings/TENCON*, vol. 2017-Decem, pp. 2061–2066, Dec. 2017, doi: 10.1109/TENCON.2017.8228200.

[36] T. Alharbi, A. Aljuhani, and Hang Liu, “Holistic DDoS mitigation using NFV,” in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, Jan. 2017, pp. 1–4, doi:

10.1109/CCWC.2017.7868480.

[37] V. F. Garcia, G. de F. Gaiardo, L. da C. Marcuzzo, R. C. Nunes, and C. R. P. dos Santos, "DeMONS: A DDoS Mitigation NFV Solution," *2018 IEEE 32nd Int. Conf. Adv. Inf. Netw. Appl.*, 2018.

[38] S. Mamolar, A. Calero, and M. Khattak, "Towards the transversal detection of DDoS network attacks in 5G multi-tenant overlay networks," doi: 10.1016/j.cose.2018.07.017.

[39] R. Kalathiripi and N. Venkatram, "Regression coefficients of traffic flow metrics (RCTFM) for DDOS defense in IoT networks," *Int. J. Commun. Syst.*, vol. 34, no. 6, Apr. 2020, doi: 10.1002/DAC.4330.

[40] S. Köksal, Y. Dalveren, B. Maiga, and A. Kara, "Distributed denial-of-service attack mitigation in network functions virtualization-based 5G networks using management and orchestration," *Int. J. Commun. Syst.*, vol. 34, no. 9, pp. 1–16, 2021, doi: 10.1002/dac.4825.

[41] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "CloudScale: Elastic resource scaling for multi-tenant cloud systems," *Proc. 2nd ACM Symp. Cloud Comput. SOCC 2011*, 2011, doi: 10.1145/2038916.2038921.

[42] Y. Ren, T. Phung-Duc, J. C. Chen, and Z. W. Yu, "Dynamic auto scaling algorithm (DASA) for 5G mobile networks," *2016 IEEE Glob. Commun. Conf. GLOBECOM 2016 - Proc.*, 2016, doi: 10.1109/GLOCOM.2016.7841759.

[43] R. S. Shariffdeen, D. T. S. P. Munasinghe, H. S. Bhathiya, U. K. J. U. Bandara, and H. M. N. D. Bandara, "Adaptive workload prediction for proactive auto scaling in PaaS systems," *Proc. 2016 Int. Conf. Cloud Comput. Technol. Appl. CloudTech 2016*, pp. 22–29, 2017, doi: 10.1109/CloudTech.2016.7847713.

[44] Y. Khettab, M. Bagaa, D. L. C. Dutra, T. Taleb, and N. Toumi, "Virtual

security as a service for 5G verticals," *IEEE Wirel. Commun. Netw. Conf. WCNC*, vol. 2018-April, pp. 1–6, Jun. 2018, doi: 10.1109/WCNC.2018.8377298.

[45] D. Sattar and A. Matrawy, "Towards Secure Slicing: Using Slice Isolation to Mitigate DDoS Attacks on 5G Core Network Slices," *2019 IEEE Conf. Commun. Netw. Secur. CNS 2019*, pp. 82–90, Jun. 2019, doi: 10.1109/CNS.2019.8802852.

[46] A. Boualouache and T. Engel, "Federated Learning-based Inter-slice Attack Detection for 5G-V2X Sliced Networks," *IEEE Veh. Technol. Conf.*, vol. 2022-Septe, 2022, doi: 10.1109/VTC2022-FALL57202.2022.10012736.

[47] S. Wijethilaka and M. Liyanage, "A Novel Network Slicing based Security-As-A-Service (SECaaS) Framework for Private 5G Networks," *2022 IEEE Latin-American Conf. Commun. LATINCOM 2022*, 2022, doi: 10.1109/LATINCOM56090.2022.10000486.

[48] S. M. A. Kazmi, L. U. Khan, N. H. Tran, and C. S. Hong, "Network Slicing for 5G and Beyond Networks," *Netw. Slicing 5G Beyond Networks*, 2019, doi: 10.1007/978-3-030-16170-5.

[49] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network Slicing in 5G: Survey and Challenges," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 94–100, May 2017, doi: 10.1109/MCOM.2017.1600951.

[50] K. Katsalis, N. Nikaein, E. Schiller, R. Favraud, and T. I. Braun, "5G Architectural Design Patterns," *2016 IEEE Int. Conf. Commun. Work. ICC 2016*, pp. 32–37, Jul. 2016, doi: 10.1109/ICCW.2016.7503760.

[51] E. Siron, *Microsoft Hyper-V Cluster Design.* Packt Publishing, 2013.

[52] J. E. Smith and R. Nair, "Virtual machines : versatile platforms for systems and processes," p. 638, 2005.

[53] T. Clark, *Storage virtualization : technologies for simplifying data storage and management*. Addison-Wesley, 2005.

[54] S. Rixner, "Network Virtualization: Breaking the Performance Barrier," *Queue*, vol. 6, no. 1, Jan. 2008, doi: 10.1145/1348583.1348592.

[55] "What is a Container? | Docker." https://www.docker.com/resources/what-container/ (accessed Jul. 15, 2022).

[56] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, and W. Zhou, "A Comparative Study of Containers and Virtual Machines in Big Data Environment," 2018.

[57] A. S. Andrew Leung, "Titus: Introducing Containers to the Netflix Cloud - ACM Queue," *ACM*, vol. Volume 15, no. issue 5, Accessed: Jul. 15, 2023. [Online]. Available: https://queue.acm.org/detail.cfm?id=3158370.

[58] W. Attaoui *et al.*, "VNF AND CONTAINER PLACEMENT: RECENT ADVANCES AND FUTURE TRENDS 1 VNF and Container Placement: Recent Advances and Future Trends," no. March, 2022, [Online]. Available: https://click.endnote.com/viewer?doi=arxiv%3A2204.00178&token=WzMyMTQ1NTMsImFyeGl2OjIyMDQuMDAxNzgiXQ.tgRNgKUZ-TOohQwGgPp90suAr_k.

[59] D. Wypiór, M. Klinkowski, and I. Michalski, "Open RAN—Radio Access Network Evolution, Benefits and Market Trends," *Appl. Sci. 2022, Vol. 12, Page 408*, vol. 12, no. 1, p. 408, Jan. 2022, doi: 10.3390/APP12010408.

[60] F. Auer, V. Lenarduzzi, M. Felderer, and D. Taibi, "From monolithic systems to Microservices: An assessment framework," *Inf. Softw.*

Technol., vol. 137, p. 106600, Sep. 2021, doi: 10.1016/J.INFSOF.2021.106600.

[61] S. Newman, *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media (US) , 2021.

[62] Fran Berman, Anthony J. G. Hey, and Geoffrey C. Fox, *Grid Computing: Making The Global Infrastructure a Reality*. 2002.

[63] David K. Rensin, *Kubernetes: Scheduling the Future at Cloud Scale*. O'Reilly Media, 2015.

[64] MerkelDirk, "Docker: lightweight Linux containers for consistent development and deployment," *Linux J.*, vol. 2014, no. 239, Mar. 2014, doi: 10.5555/2600239.2600241.

[65] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource allocation algorithms for virtualized service hosting platforms," *J. Parallel Distrib. Comput.*, vol. 70, no. 9, pp. 962–974, Sep. 2010, doi: 10.1016/J.JPDC.2010.05.006.

[66] Christopher M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[67] by J. Ross Quinlan, M. Kaufmann Publishers, and S. L. Salzberg, "C4.5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993," *Mach. Learn. 1994 163*, vol. 16, no. 3, pp. 235–240, Sep. 1994, doi: 10.1007/BF00993309.

[68] J. R. Quinlan, "Simplifying decision trees," *Int. J. Man. Mach. Stud.*, vol. 27, no. 3, pp. 221–234, Sep. 1987, doi: 10.1016/S0020-7373(87)80053-6.

[69] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001, doi: 10.1023/A:1010933404324/METRICS.

[70] W. W. Cohen, “Fast Effective Rule Induction,” *Proc. 12th Int. Conf. Mach. Learn. ICML 1995*, pp. 115–123, Jan. 1995, doi: 10.1016/B978-1-55860-377-6.50023-2.

[71] H. Berger, D. Merkl, and M. Dittenbach, “Exploiting partial decision trees for feature subset selection in e-mail categorization,” *Proc. ACM Symp. Appl. Comput.*, vol. 2, pp. 1105–1109, 2006, doi: 10.1145/1141277.1141536.

[72] M. A. Nielsen, “Neural Networks and Deep Learning.” Determination Press, 2015, Accessed: Aug. 10, 2023. [Online]. Available: http://neuralnetworksanddeeplearning.com.

[73] H. Abusamra, “A comparative study of feature selection and classification methods for gene expression data of glioma,” *Heba Abus. / Procedia Comput. Sci.*, vol. 23, pp. 5–14, 2013, doi: 10.1016/j.procs.2013.10.003.

[74] F. Lau, S. H. Rubin, M. H. Smith, and L. Trajković, “Distributed denial of service attacks,” *Proc. IEEE Int. Conf. Syst. Man Cybern.*, vol. 3, pp. 2275–2280, 2000, doi: 10.1109/ICSMC.2000.886455.

[75] D. Kumar, B. Jugal, K. Kalita, and D. Attacks, “DDoS Attacks : Evolution, Detection, Prevention, Reaction, and Tolerance,” *DDoS Attacks*, Apr. 2016, doi: 10.1201/B20614.

[76] N. Provos and T. Holz, *Virtual honeypots : from botnet tracking to intrusion detection*. Addison-Wesley, 2007.

[77] C. C. Zou and R. Cunningham, “Honeypot-aware advanced botnet construction and maintenance,” *Proc. Int. Conf. Dependable Syst. Networks*, vol. 2006, pp. 199–208, 2006, doi: 10.1109/DSN.2006.38.

[78] L. Spitzner, “The honeynet project: Trapping the hackers,” *IEEE Secur. Priv.*, vol. 1, no. 2, pp. 15–23, 2003, doi:

10.1109/MSECP.2003.1193207.

[79] S. Axelsson, “Intrusion Detection Systems: A Survey and Taxonomy,” 2000.

[80] P. Natesan, P. Balasubramanie, and G. Gowrison, “Improving the Attack Detection Rate in Network Intrusion Detection using Adaboost Algorithm,” *J. Comput. Sci.*, vol. 8, no. 7, pp. 1041–1048, May 2012, doi: 10.3844/JCSSP.2012.1041.1048.

[81] “Concepts | Kubernetes.” https://kubernetes.io/docs/concepts/ (accessed Jul. 15, 2023).

[82] “Calico Documentation | Calico Documentation.” https://docs.tigera.io/ (accessed Aug. 10, 2022).

[83] “Overview | Prometheus.” https://prometheus.io/docs/introduction/overview/ (accessed Jul. 15, 2022).

[84] “Grafana OSS | Visualization and dashboarding technology.” https://grafana.com/oss/ (accessed Aug. 10, 2023).

[85] “The Official YAML Web Site.” https://yaml.org/ (accessed Aug. 10, 2022).

[86] “Kubectl Reference Docs.” https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands (accessed Aug. 12, 2023).

[87] “Common Data Set | UCLA Academic Planning and Budget.” https://apb.ucla.edu/campus-statistics/common-data-set (accessed Aug. 11, 2022).

[88] Kurniabudi, D. Stiawan, Darmawijoyo, M. Y. Bin Idris, A. M. Bamhdi, and R. Budiarto, “CICIDS-2017 Dataset Feature Analysis With

Information Gain for Anomaly Detection," *IEEE Access*, vol. 8, pp. 132911–132921, 2020, doi: 10.1109/ACCESS.2020.3009843.

[89] S. Daneshgadeh Çakmakçı, T. Kemmerich, T. Ahmed, and N. Baykal, "Online DDoS attack detection using Mahalanobis distance and Kernel-based learning algorithm," *J. Netw. Comput. Appl.*, vol. 168, p. 102756, 2020, doi: https://doi.org/10.1016/j.jnca.2020.102756.

[90] K. Kurniabudi, D. Stiawan, D. Darmawijoyo, M. Y. Bin Idris, B. Kerim, and R. Budiarto, "Important Features of CICIDS-2017 Dataset For Anomaly Detection in High Dimension and Imbalanced Class Dataset," *Indones. J. Electr. Eng. Informatics*, vol. 9, no. 2, pp. 498–511, 2021.

[91] Y. M. Swe, P. P. Aung, and A. S. Hlaing, "A Slow DDoS Attack Detection Mechanism using Feature Weighing and Ranking," *Proc. 11th Annu. Int. Conf. Ind. Eng. Oper. Manag. Singapore*, 2021.

[92] M. I. Kareem and M. N. Jasim, "Fast and accurate classifying model for denial-of-service attacks by using machine learning," *Bull. Electr. Eng. Informatics*, vol. 11, no. 3, pp. 1742–1751, Jun. 2022, doi: 10.11591/EEI.V11I3.3688.

[93] "Report on DDoS attacks in Q3 2022 | Securelist." https://securelist.com/ddos-report-q3-2022/107860/ (accessed Aug. 14, 2022).

[94] "Kaspersky Q4 2021 DDoS attack report | Securelist." https://securelist.com/ddos-attacks-in-q4-2021/105784/ (accessed Aug. 14, 2022).

[95] "DDoS attacks in Q4 2020 | Securelist." https://securelist.com/ddos-attacks-in-q4-2020/100650/ (accessed Aug. 14, 2022).

[96] "DDoS attacks in Q4 2019 | Securelist." https://securelist.com/ddos-report-q4-2019/96154/ (accessed Aug. 14, 2022).

[97] E. H. Beni, E. Truyen, B. Lagaisse, W. Joosen, and J. Dieltjens, “Reducing cold starts during elastic scaling of containers in kubernetes,” *Proc. ACM Symp. Appl. Comput.*, pp. 60–68, Mar. 2021, doi: 10.1145/3412841.3441887.

[98] C. Carrión, “Kubernetes Scheduling: Taxonomy, Ongoing Issues and Challenges,” *ACM Comput. Surv.*, vol. 55, no. 7, Dec. 2022, doi: 10.1145/3539606.

الخلاصة

تمثل شبكات الجيل القادم (NGN) تطور وتقارب مختلف تقنيات وخدمات وشبكات الاتصالات لتلبية المتطلبات المتزايدة للمجتمع. وتهدف شبكات الجيل القادم إلى توفير قدرات معززة وأداء محسن ومرونة أكبر مقارنة بالشبكات التقليدية. إحدى السمات الأساسية لشبكات الجيل التالي هي قدرتها على دعم شبكات الوصول المتعدد، مما يسمح للمستخدمين بالاتصال عبر تقنيات وصول مختلفة مثل الشبكات السلكية واللاسلكية والمتنقلة. يتيح ذلك الاتصال في كل مكان والتنقل السلس، وتمكين المستخدمين من الوصول إلى الخدمات في أي وقت وفي أي مكان وعلى أي جهاز ولكن هذه الميزة لها تأثير أمني يمكن أن يقلل من أداء الشبكة ويندد بسلامة البيانات المنقولة. تمثل هجمات رفض الخدمة الموزعة (DDoS) تهديدًا كبيرًا لشبكات الجيل التالي. يطغى المهاجمون على موارد الشبكة بكمية هائلة من حركة المرور من مصادر متعددة.

اقترحت هذه الدراسة بنية تحتية قابلة للتطوير يمكنها تخصيص الموارد ديناميكيًا وضبط تدفق حركة المرور للتخفيف من تأثير هجمات DDoS .تم استخدام خمس خوارزميات مختلفة للعثور على الخوارزمية الأكثر دقة والأسرع فيما يتعلق بوقت التدريب. في الاختبار، اثبتت خوارزمية الـ RF بأنها اعلى دقة من بين الخوارزميات فيما كانت الـ J48 الخوارزمية الاسرع في وقت التدريب. بعد ذلك تم دمج الخوارزميتين معًا للحصول على نموذج مكدس جديد حصل على دقة ٩٩,٩٩٤٤٪ في وقت الاختبار الـ offline.

الهدف الأساسي للنموذج المكدس هو تطوير مصنف DDoS باستخدام خوارزمية التعلم الآلي ونشره داخل حاوية Docker في بيئة افتراضية للتعامل بكفاءة مع الهجمات المحتملة، تم توظيف منسق للإشراف على عملية القياس، وإنشاء مثيلات إضافية للمصنف ديناميكيًا. يعتمد عدد المصنفات التي تم إنشاء مثيل لها على الموارد المتاحة (وحدة المعالجة المركزية وذاكرة الوصول العشوائي) وكثافة الهجوم. وتشير النتائج إلى أن النهج المقترح يمنع بشكل فعال من هجوم DDoS بدقة تتراوح بين ٩٧% إلى ٩٥,١% حسب شدة الهجوم والموارد المتاحة.

وزارة التعليم العالي والبحث العلمي

جامعـــــــــــــة بابـــــــــل

كليـــــة تكنولوجيا المعلومـــــات

قسم شبكـــــات المعلومـــــات

نظام تعلم آله وتخصيص ديناميكي لتحسين امنية شبكات الجيل القادم ضد هجمات الـ DDoS

أطروحة مقدمة الى مجلس كلية تكنولوجيا المعلومات في جامعة بابل وهي جزء من متطلبات الحصول على شهادة الدكتوراه فلسفة في تكنولوجيا المعلومات / شبكات المعلومات

مقدمة من قبل

محمد جواد كاظم عبود

باشراف

الأستاذ الدكتور غسان حميد عبد المجيد

2023A.D. **1445 A.H.**