ROYAUME DU MAROC
Université Mohammed V - RABAT -
جامعة محمد الخامس – الرباط –
Faculté des sciences
كلية العلوم
CENTRE D'ETUDES DOCTORALES - SCIENCES ET TECHNOLOGIES

N° d'ordre 4071

THESE

en vue de l'obtention du : **DOCTORAT**

Structure de Recherche : Intelligent Processing and Security of Systems
Discipline : Sciences et Technologies
Spécialité : Sécurité des réseaux et Intelligence Artificielle

Présentée et Soutenue le : 12/04/2025

par :

Hicham YZZOGH

Using machine learning-based techniques to enhance software-defined network security

Devant le JURY:

Soumia ZITI	PES, Faculté des Sciences, Université Mohammed V, Rabat	Présidente
Abderrahim Ait WAKRIME	MCH, Faculté des Sciences, Université Mohammed V, Rabat	Examinateur/Rapporteur
Youssef BADDI	MCH, Ecole Supérieure de Technologie, Université Chouaib Doukkalli, El Jadida	Examinateur/Rapporteur
Cherkaoui LEGHRIS	PES, Faculté des Sciences et Techniques, Université Hassan II, Casablanca	Examinateur/Rapporteur
Ahmed EL-YAHYAOUI	MCH, Faculté des Sciences, Université Mohammed V, Rabat	Examinateur
Ahmed LEKSSAYS	Researcher, Qatar Computing Research Institute, Qatar	Invité
Hafssa BENABOUD	PES, Faculté des Sciences, Université Mohammed V, Rabat	Directrice de thèse

Année Universitaire : 2024-2025

Dedication

To the cherished memory of my dear father, though you are no longer with us, your wisdom, integrity, and the values you instilled in me continue to shape who I am. Your voice echoes in my heart, encouraging me to always strive for excellence. This achievement is a tribute to your everlasting influence on my life.

To my beloved mother, your love is the anchor of my soul. Your tireless prayers, your endless sacrifices, and your unwavering faith in me have been my greatest source of motivation. You have been my strength in moments of doubt and my comfort in times of struggle. This accomplishment is as much yours as it is mine.

To my beloved wife, your affection has been my haven, your patience my pillar, and your belief in me my driving force. Thank you for the countless sacrifices and the enduring support that helped make this dream a reality. I am forever grateful for your presence by my side.

To my wonderful children, Ilyass, Yousra, and Aya, your smiles have brightened my darkest days, and your love has filled my heart with purpose. May this work be a source of inspiration to you, and may it remind you that with passion, hard work, and resilience, any dream is within reach.

To my brothers and sister, your unwavering support, encouragement, and presence through every phase of this journey have meant the world to me. Thank you for always being there.

To my entire family, and to all the friends and well-wishers who have stood by me with kindness and encouragement, I share this success with you. Your belief in me helped carry me through, and for that, I am eternally thankful.

Hicham YZZOGH

Acknowledgement

The research conducted in this thesis was carried out within the Department of Computer Science at the Faculty of Sciences in Rabat, under the supervision of Professor **Hafssa BENABOUD**.

First of all, I would like to express my deep gratitude to my thesis supervisor, Madam **Hafssa BENABOUD**, Professor of Higher Education at the Faculty of Sciences, Mohammed V University of Rabat, for her guidance, patience, great availability, and for her human and scientific qualities. I sincerely thank her for all the assistance she provided me throughout the completion of this work.

I would also like to express my sincere gratitude to Madam **Soumia ZITI**, Professor of Higher Education at the Faculty of Sciences, Mohammed V University of Rabat, for her availability and for the honor she gave me by agreeing to chair my defense.

I would like to express my profound gratitude to Mr. **Abderrahim AIT WAKRIME**, Associate Professor at the Faculty of Sciences, Mohammed V University of Rabat, for the rigor and interest he dedicated to evaluating this work as a rapporteur.

I would also like to express my deep gratitude to Mr. **Youssef BADDI**, Associate Professor at the Higher School of Technology, Chouaib Doukkali University of El Jadida, for the rigor he demonstrated and for his commitment as a rapporteur.

I would like to warmly thank Mr. **Cherkaoui LEGHRIS**, Professor of Higher Education at the Faculty of Sciences and Techniques of Mohammedia, Hassan II University, for the attention he gave to my work and for the honor he gave me by participating in the evaluation as a rapporteur.

I am deeply grateful to Mr. **Ahmed EL-YAHYAOUI**, Associate Professor at the Faculty of Sciences, Mohammed V University of Rabat, for the rigor and interest he showed in evaluating this work as an examiner.

I would like to express my gratitude to Mr. **Ahmed LEKSSAYS**, Security Researcher at the Qatar Computing Research Institute, for the honor he gave me by accepting to participate as an invited guest at this defense. His presence is a testament to his interest in this work, and I am deeply grateful to him for it.

Finally, I cannot forget all those who contributed, directly or indirectly, to the completion of this work.

Résumé

L'évolution rapide des services et des applications numériques a augmenté la complexité des réseaux modernes. Les réseaux définis par logiciel (SDN) offrent une gestion centralisée et une programmabilité accrue en séparant le contrôle du transfert de données. Cependant, la centralisation du plan de contrôle expose les SDN aux attaques DDoS et à d'autres menaces avancées, compromettant ainsi la fiabilité des services.

Cette thèse explore l'application du Machine Learning pour améliorer la sécurité des SDN via la classification du trafic et la détection d'intrusions. Les méthodes traditionnelles, comme le filtrage par port et l'inspection de paquets, sont inadéquates face au volume élevé, au trafic chiffré et à la nature dynamique des réseaux modernes. L'apprentissage automatique permet une identification plus précise des cybermenaces émergentes.

Trois approches sont étudiées : la première combine le clustering K-Means avec Naïve Bayes pour détecter les attaques DDoS par inondation. Les deux autres concernent la classification du trafic : l'une combine K-Means et Word2Vec avec un réseau de neurones, tandis que l'autre applique l'extraction de caractéristiques avec Oriented FAST and Rotated BRIEF (ORB), suivie d'une classification par réseau de neurones.

Mots Clés: Réseaux définis par logiciel, Apprentissage automatique, Sécurité, Attaques DDoS par inondation, K-Means, Naïve Bayes, Word2Vec, Réseau de neurones, Réseau de neurones convolutionnel, Oriented FAST and Rotated BRIEF, Classification du trafic.

Abstract

The rapid evolution of digital services and applications has significantly increased the complexity of modern networks, challenging traditional architectures to adapt to dynamic demands. Software-Defined Networking (SDN) introduces a paradigm shift by separating the data plane and control plane, providing centralized management and enhanced programmability. While this flexibility enhances scalability and responsiveness, the centralization of SDN's control plane introduces critical vulnerabilities, including exposure to Distributed Denial-of-Service (DDoS) attacks and other sophisticated threats. Ensuring effective security in SDN environments is crucial to maintaining the integrity and reliability of essential services.

This thesis addresses the security challenges of SDN through the application of machine learning techniques for traffic classification and intrusion detection. Traditional methods, such as port-based filtering and deep packet inspection, are inadequate for the high volume, encrypted traffic, and dynamic nature of modern networks. Machine learning presents a promising alternative, offering adaptability and precision in identifying nuanced traffic patterns and detecting evolving cyber threats.

The research explores three distinct approaches to enhancing SDN security. The first approach combines K-Means clustering with Naïve Bayes for Flooding DDoS detection. The two other approaches focus on traffic classification: the first integrates K-Means and Word2Vec with a neural network, while the second applies feature extraction using Oriented FAST and Rotated BRIEF (ORB), followed by a neural network. Each approach is evaluated for its effectiveness in addressing SDN's unique security challenges. The findings contribute to advancing SDN security frameworks, providing insights into scalable, adaptive methods for protecting these next-generation networks against emerging threats.

Keywords: Software-Defined Networking, Machine Learning, Security, Flooding DDoS Attacks, K-Means, Naïve Bayes, Word2Vec, Neural Network, Convolutional Neural Network, Oriented FAST and Rotated BRIEF, Traffic Classification.

ملخص

ساهم التطور السريع في الخدمات الرقمية والتطبيقات في زيادة تعقيد الشبكات الحديثة بشكل كبير، حتى أصبحت البُنى التقليدية عاجزة عن مواكبة المتطلبات الديناميكية والمتغيرة باستمرار. يقدم مفهوم "الشبكات المعرفة بالبرمجيات" (SDN) حلاً عملياً لهذه المشكلة من خلال فصل طبقة البيانات عن طبقة التحكم، مما يتيح إدارة مركزية وقدرات برمجية محسّنة. ورغم أن هذه المرونة تعزز من قابلية التوسع وسرعة الاستجابة، فإن تركيز التحكم في جهة واحدة يُفتح أمام الشبكة ثغرات أمنية خطيرة، أبرزها هجمات الحرمان من الخدمة الموزّعة (DDoS) والتهديدات السيبرانية المتطورة الأخرى. ومن ثم، أصبح تحقيق أمان فعّال في بيئات SDN أمراً أساسياً للحفاظ على سلامة وموثوقية الخدمات الحيوية.

تتناول هذه الأطروحة تحديات الأمان في شبكات SDN من خلال توظيف تقنيات التعلم الآلي في تصنيف حركة المرور والكشف عن التسللات، حيث لم تعد الأساليب التقليدية مثل فلترة المنافذ وفحص الحزم العميق قادرة على التعامل مع حجم البيانات الهائل والطبيعة المشفرة والديناميكية لحركة الشبكة اليوم. يوفر التعلم الآلي بديلاً واعداً، إذ يتمتع بقدرة عالية على التكيف مع التغيرات ودقة في اكتشاف أنماط حركة المرور المعقدة ورصد التهديدات الناشئة بشكل أسرع وأكثر فعالية.

يستعرض البحث ثلاث مقاربات متميزة لتعزيز أمان شبكات SDN. المقاربة الأولى تجمع بين عدة نماذج K-Means ومصنف Naïve Bayes لاكتشاف هجمات DDoS الموجهة عبر الفيض. المقاربتان الأخريان تركزان على تصنيف حركة المرور: الأولى تدمج بين K-Means وWord2Vec مع شبكة عصبية، بينما الثانية تطبق استخراج السمات باستخدام تقنية Oriented FAST and Rotated BRIEF (ORB)، تليها شبكة عصبية. تم تقييم كل مقاربة من حيث فعاليتها في معالجة التحديات الأمنية الفريدة لشبكات SDN. تساهم النتائج في تطوير أطر أمان شبكات SDN، وتقديم رؤى حول طرق قابلة للتوسع والتكيف لحماية هذه الشبكات المتقدمة ضد التهديدات الناشئة.

الكلمات الأساسية :الشبكات المعرفة بالبرمجيات، التعلم الآلي، الأمان، هجمات DDoS الموجهة عبر الفيض، K-Means، Naïve Bayes، Word2Vec، الشبكة العصبية، الشبكة العصبية التلافيفية، ORB، تصنيف حركة المرور.

Résumé Détaillé

L'évolution rapide des services numériques a transformé les réseaux modernes, rendant les architectures traditionnelles inadaptées aux exigences dynamiques actuelles. Les réseaux définis par logiciel (SDN) proposent une alternative en dissociant le plan de contrôle du plan de données, ce qui permet une gestion centralisée et une programmabilité accrue. Cette flexibilité, bien que bénéfique pour des domaines comme l'IoT ou le cloud computing, introduit aussi de nouvelles vulnérabilités, notamment une dépendance accrue au plan de contrôle et une exposition plus importante aux attaques telles que les DDoS. De plus, les approches classiques de sécurité, fondées sur des règles statiques ou des signatures connues, ne parviennent pas à suivre l'évolution rapide des cybermenaces, en particulier dans des environnements où le trafic est chiffré et fortement dynamique.

Cette thèse explore trois approches basées sur l'apprentissage automatique pour améliorer la sécurité dans les environnements SDN. La première approche consiste à entraîner un modèle combinant plusieurs modèles K-Means et un classifieur Naïve Bayes Gaussien. Pour ce faire, des modèles K-Means sont entraînés en parallèle et de manière indépendante, chaque modèle étant dédié à une caractéristique de l'ensemble de données. Ces modèles sont ensuite utilisés pour assigner des clusters aux données des caractéristiques sur lesquelles ils ont été entraînés, et ces clusters servent d'entrée pour entraîner le classifieur Naïve Bayes Gaussien. Cette méthode permet de classifier les flux avec un faible coût computationnel, atteignant une précision de 99,98 % sur InSDN.

La deuxième approche propose une technique de classification du trafic où les clusters assignés en utilisant les modèles K-Means sont transformés en vecteurs Word2Vec. Ensuite, chaque flux est converti en une représentation numérique en calculant la moyenne des vecteurs Word2Vec correspondants. Enfin, ces représentations numériques sont utilisées comme entrées pour un réseau de neurones. Cette combinaison permet de capturer des relations complexes au sein du flux, avec une précision atteignant jusqu'à 99,97 % sur InSDN.

Dans la troisième approche, l'extraction des caractéristiques est effectuée par ORB, en alternative aux CNN, qui sont plus coûteux en termes de calcul. Cette méthode repose sur l'extraction de caractéristiques depuis des images générées à partir des données de flux, avec l'objectif de renforcer la sécurité des SDN et de trouver une alternative aux CNN. Trois méthodes sont utilisées pour générer

les images : la conversion de flux en diagramme à barres, l'IGTD avec la distance Euclidienne et l'IGTD avec la distance Manhattan. Les descripteurs extraits par ORB sont ensuite utilisés comme features d'un réseau de neurones chargé de classer le trafic. Les expériences menées sur les ensembles de données InSDN et CIC-DDoS2019 montrent que cette approche permet une classification précise (jusqu'à 99,86,% d'accuracy sur CIC-DDoS2019), avec des temps d'entraînement réduits et une meilleure adaptabilité aux contraintes de traitement en temps réel par rapport aux CNN.

Ce travail présente des solutions innovantes basées sur les caractéristiques pour améliorer la classification du trafic et la détection des intrusions, tout en prenant en compte la surcharge computationnelle. Nous avons également intégré des techniques de traitement du langage naturel et de vision par ordinateur.

Publications

This thesis is partially based on the following peer-reviewed publications:

International Journals

- **H. Yzzogh** and H. Benaboud, ”A Comprehensive Overview of Machine Learning for Intrusion Detection in Software-Defined Networking,” *Innovations Syst Softw Eng*, 2025, doi: 10.1007/s11334-025-00604-6.
- **H. Yzzogh** and H. Benaboud, “Flooding distributed denial of service detection in software-defined networking using k-means and naïve Bayes,” *IJECE*, vol. 15, no. 1, pp. 817–826, Feb. 2025, doi: 10.11591/ijece.v15i1.pp817-826.
- **H. Yzzogh** and H. Benaboud, “Enhancing SDN security with a feature-based approach using multiple k-means, Word2Vec, and neural network,” *Bulletin EEI*, vol. 14, no. 2, pp. 1456–1467, Apr. 2025, doi: 10.11591/eei.v14i2.8834.
- **H. Yzzogh** and H. Benaboud, “Evaluating ORB and SIFT With Neural Network as Alternatives to CNN for Traffic Classification in SDN Environments,” *IEEE Access*, vol. 13, pp. 51484–51498, 2025, doi: 10.1109/ACCESS.2025.3553449.

International Conference

- **H. Yzzogh** and H. Benaboud, ”Using SDN to Enhance Load Balancing in Cloud Computing: An Overview and Future Directions,” 2023 6th International Conference on Advanced Communication Technologies and Networking (CommNet), Rabat, Morocco, 2023, pp. 1-6, doi: 10.1109/CommNet60167.2023.10365294.

List of Figures

List of Tables

List of Abbreviations

ACLs	Access Control Lists
ACC	Accuracy
AdaBoost	Adaptive Boosting
AE	AutoEncoders
ANN	Artificial Neural Network
AUC	Area Under the Curve
BFA	Brute Force Attack
BRIEF	Binary Robust Independent Elementary Features
BT	Bagging Tree
CART	Classification and Regression Trees
CatBoost	Categorical Boosting
CNNs	Convolutional Neural Networks
CSVC	C-Support Vector Machine
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DDoS	Distributed Denial of Service
DNN	Deep Neural Network
DNS	Domain Name System
DoG	Difference of Gaussians
DT	Decision Tree
FNN	Feedforward Neural Network
FM	Factorization Machines
FAST	Features from Accelerated Segment Test
FNR	False Negative Rate
FPR	False Positive Rate
GA	Genetic Algorithm
GBC	Gradient Boosting Classifier

GLM	Generalized Linear Model
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
IDS	Intrusion Detection System
IGTD	Image Generator for Tabular Data
IPS	Intrusion Prevention System
IoT	Internet of Things
KNN	K-Nearest Neighbors
KPCA	Kernel Principal Component Analysis
LDA	Linear Discriminant Analysis
LOA	Lion Optimization Algorithm
LOIC	Low Orbit Ion Cannon
LR-DDoS	Low Rate DDoS
LSTM	Long Short-Term Memory
MiTM	Man-in-the-Middle
ML	Machine Learning
MLP	Multi-Layer Perceptron
NB	Naïve Bayes
NN	Neural Network
NLP	Natural Language Processing
ONF	Open Networking Foundation
ORB	Oriented FAST and Rotated BRIEF
PCA	Principal Component Analysis
QoS	Quality of Service
QDA	Quadratic Discriminant Analysis
RBM	Restricted Boltzmann Machines
RFE	Recursive Feature Elimination
RF	Random Forest
RNN	Recurrent Neural Network
RSBV	Range Supported Bit Vector
SAE	Stacked Auto-Encoder
SDN	Software-Defined Networking
SENS	Sensitivity

SIFT	Scale-Invariant Feature Transform
SPEC	Specificity
SVM	Support Vector Machine
SVD	Singular Value Decomposition
t-SNE	t-Distributed Stochastic Neighbor Embedding
TLS	Transport Layer Security
TPR	True Positive Rate
ViT	Vision Transformer
VLAN	Virtual Local Area Network
WFL	Weighted Federated Learning
Word2Vec	Word to Vector
XGBoost	eXtreme Gradient Boosting

Contents

General Introduction

Traditional network architectures, originally designed for static and hardware-dependent environments, face significant challenges in meeting the dynamic and flexible demands of modern services. Software-Defined Networking (SDN) addresses these challenges by decoupling the control plane, which handles decision-making, from the data plane, which manages data forwarding. This separation provides organizations with centralized management capabilities and programmability, enabling network administrators to configure, optimize, and scale resources with greater flexibility and efficiency.

SDN's agility is especially beneficial in scenarios requiring real-time adaptability, such as the Internet of Things (IoT), cloud computing, and data analytics. In these contexts, the ability to respond quickly to changing conditions is critical for success. However, this adaptability relies on centralized control, which, while enabling dynamic management, also introduces risks, including a single point of failure and increased vulnerability to cyber threats.

These vulnerabilities make securing SDN environments a critical priority. A compromise of the control plane can lead to widespread and cascading disruptions across the network. To address these risks, implementing robust and scalable security mechanisms is essential to ensuring the reliability, availability, and confidentiality of SDN systems. In this evolving landscape, adopting advanced security measures is imperative for organizations to safeguard their SDN implementations and effectively mitigate potential threats.

Given these evolving challenges in SDN security, it becomes clear that traditional network security methods, originally designed for static and hardware-dependent architectures, struggle to address the dynamic and high-volume nature of modern SDN environments. Approaches such as port-based filtering and static rule sets, which rely on predefined policies, are inadequate for the task. These methods fail to keep up with emerging threats and the dynamic demands of SDN.

Furthermore, traditional security mechanisms face additional challenges in handling encrypted and high-speed traffic, both of which are increasingly prevalent in SDN environments. As encryption becomes more widely adopted for privacy and performance enhancements, conventional approaches often lack the capability to analyze traffic payloads or maintain sufficient visibility into data streams. This limited visibility introduces significant security risks, making it harder to detect emerging threats and

respond in real time. Moreover, the integration of advanced technologies like virtualization and automation only complicates security enforcement, creating even more opportunities for vulnerabilities to be exploited.

To address these challenges, machine learning (ML)-based approaches offer a more adaptive and scalable solution. Unlike traditional security mechanisms, ML can analyze network traffic patterns in real time without relying on predefined signatures. By continuously learning from evolving network behaviors, ML can dynamically detect anomalies, uncover hidden attack patterns, and strengthen SDN security resilience.

The effectiveness of ML in SDN security, however, hinges on its ability to process traffic efficiently while maintaining real-time performance. The integration of ML into SDN frameworks brings challenges related to computational overhead, model scalability, and the need for rapid inference. Overcoming these obstacles is essential to ensure that ML-based solutions are both practical and effective for real-world SDN deployments.

In this context, this research focuses on leveraging machine learning to design and develop innovative Intrusion Detection Systems (IDS) capable of effectively identifying a wide range of security threats, such as DoS, DDoS, and Brute Force Attacks (BFA), in SDN environments. The centralized control plane introduces unique vulnerabilities that require advanced, adaptive solutions to safeguard against potential attacks.

The scope of this research begins with a thorough analysis of SDN architecture and its associated security risks [1]. This includes evaluating components like the control and data planes, as well as protocols such as OpenFlow, to highlight the inherent vulnerabilities in SDN systems. Based on this analysis, the research proposes three distinct machine learning-based approaches to enhance security, and includes a comparative evaluation of their strengths and limitations to determine the most effective solution for addressing the challenges of SDN security.

The key contributions of this research are as follows: The first contribution [2] focuses on detecting flooding DDoS attacks in SDN using one-dimensional K-Means clustering and Naïve Bayes classification. This method demonstrates high accuracy, achieving 99.98% on the InSDN dataset and 99.70% on CIC-DDoS2017. The second contribution [3] introduces a feature-engineering-based approach using multiple K-Means, Word2Vec, and a neural network to improve traffic classification and intrusion detection, achieving an accuracy of 99.97% on InSDN and 98.65% on CIC-DDoS2019. The novelty of this method lies in the use of Word2Vec, which is designed to learn from sequences of tokens by capturing the context in which 'words' appear. In our use case, it identifies patterns where certain clusters frequently occur together, reflecting specific traffic behaviors such as attacks or normal activity. Finally, the third contribution [4] evaluates the use of ORB and SIFT as alternatives to Convolutional Neural

Networks (CNNs) for traffic classification. The ORB model achieves 97.14% accuracy on InSDN and 99.86% on CIC-DDoS2019 using our unique flow-to-bar chart conversions, with significantly reduced computational overhead compared to the CNN model. All experiments in this thesis were conducted on a system equipped with an Intel® Core™ i7-6820HQ processor (2.70 GHz), 32 GB of RAM, running Windows 10.

The thesis is organized as follows: Chapter 1 provides an overview of SDN, its architecture, and security challenges, setting the stage for the role of machine learning in this domain. Chapter 2 introduces the first proposed approach, detailing its methodology, experimental setup, results, and analysis. Chapters 3 and 4 describe the remaining two approaches, including their design, performance evaluation, and outcomes. Chapter 5 offers a synthesis of the three approaches, discussing their strengths and limitations.

Chapter 1

Challenges in Securing SDN Networks Using Machine Learning: A Comprehensive Review

With the expansion of modern networks driven by the Internet of Things, the Industrial Internet, and smart cities, security challenges are becoming increasingly complex. Traditional security measures, such as rule-based firewalls and signature-based intrusion detection systems, are proving inadequate against sophisticated cyber threats, highlighting the need for adaptive and intelligent security mechanisms. To effectively counter these evolving threats, network architectures must not only defend against known attacks but also anticipate and adapt to emerging security risks.

One of the most promising advancements in addressing these challenges is Software-Defined Networking, which enhances network efficiency and flexibility through centralized control and programmability. However, this architectural transformation also introduces new security concerns. Unlike traditional networks, where the control and data planes are integrated, SDN separates these functions, expanding the attack surface and introducing new vulnerabilities. Exploits in protocols such as OpenFlow can allow adversaries to manipulate traffic flows, disrupt services, and launch DoS and DDoS attacks. Consequently, SDN requires adaptive security mechanisms capable of real-time threat detection and mitigation to ensure network resilience in increasingly dynamic and hostile environments.

To strengthen SDN security, Machine Learning has emerged as a powerful tool for network defense. By leveraging data-driven models, ML enables intelligent threat detection, anomaly identification, and adaptive defense strategies. The integration of ML into SDN security frameworks has gained significant attention, as it facilitates real-time network behavior analysis and enhances the detection of sophisticated cyber threats. ML-based Intrusion Detection Systems, for instance, offer a proactive security approach by continuously learning from evolving attack patterns and improving response mechanisms. However, despite these advantages, ML-based security solutions also introduce new challenges, including the risk of adversarial attacks, where manipulated input data can evade detection. Additionally, the computational overhead associated with large-scale ML deployments poses concerns for system performance and scalability.

In the upcoming section, we provide a detailed analysis of SDN architecture, examining its application, control, and data planes and highlighting key security challenges.

1.1 SDN Architecture and Related Security Challenges

1.1.1 SDN Architecture

The Software-Defined Networking architecture consists of three fundamental planes, each playing a crucial role in network management and operation: the application plane, which hosts network applications that define policies and request services; the control plane, responsible for decision-making and data flow; and the data plane, which handles data transmission. Together, these planes ensure efficient network performance. Figure 1.1 illustrates their interaction, highlighting how they collaborate to create a streamlined and adaptable network environment.

The application plane serves as the highest layer in the SDN architecture, housing network applications that provide functionalities such as traffic engineering, security enforcement, and quality of service (QoS) management. These applications interact with the SDN controller through the northbound interface to define high-level policies and influence network behavior. By leveraging programmability, the application plane enables dynamic network management and automation, ensuring adaptability to changing requirements.

The control plane, often referred to as the ”brain” of the network, orchestrates and manages SDN elements. It enables efficient data flow and decision-making, adapting to evolving conditions. This plane comprises one or more SDN controllers, which serve as centralized units that analyze real-time network conditions, interpret application plane policies, and determine optimal routing paths. Based on these decisions, they generate forwarding rules that guide data movement. Communication with the data plane occurs through southbound APIs, with OpenFlow being a widely used protocol for enforcing these rules and managing network resources.

The data plane, or forwarding plane, handles the actual transmission of data, ensuring seamless communication between devices. It includes physical and virtual switches, routers, and load balancers, which forward traffic based on rules defined by the control plane. For instance, OpenFlow-compatible switches maintain flow tables that store control plane rules, enabling precise and efficient routing.

For a deeper understanding of SDN architecture, RFC7426 [5] provides comprehensive details, while RFC7276 [6] offers an overview of Operations, Administration, and Maintenance (OAM) tools available for SDN systems.

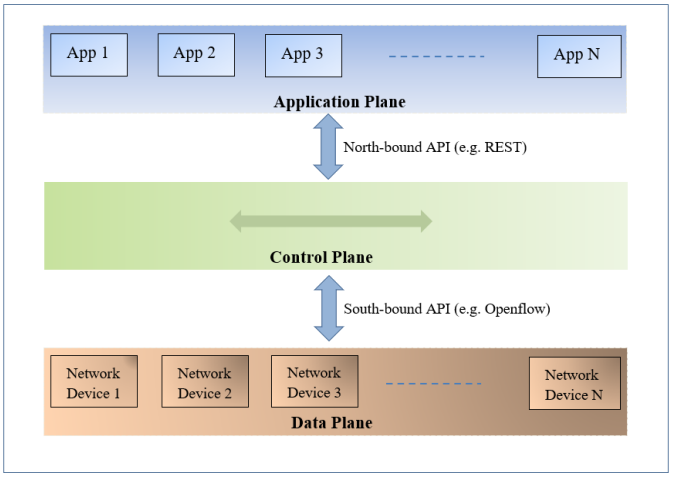


Figure 1.1: SDN architecture overview

1.1.2 OpenFlow Protocol

OpenFlow is a fundamental component of SDN, serving as the primary southbound interface between the control plane and the data plane. It provides a standardized mechanism for the SDN controller to configure flow table entries in network devices such as switches and routers. Flow tables contain rules that determine how incoming packets are processed and routed based on criteria such as source and destination addresses, port numbers, and protocol types. By dynamically updating these rules, OpenFlow enables efficient traffic management and optimized routing.

Managed by the Open Networking Foundation (ONF), OpenFlow allows the SDN controller to translate high-level policies into forwarding rules that can be applied in real time. This capability enables precise control over network traffic, facilitating adaptive routing, QoS enforcement, and security measures. For example, OpenFlow can reroute traffic to avoid congestion and implement security policies dynamically.

OpenFlow also allows switches to notify the controller when packets do not match any existing rules in the flow table. In response, the controller analyzes the unmatched packets and issues new forwarding instructions, ensuring adaptability to changing network conditions. This mechanism supports dynamic reconfiguration without manual intervention, making it valuable for handling new devices and evolving traffic patterns.

Since its introduction, OpenFlow has evolved through multiple versions, improving scalability, flexibility, and functionality. Later versions introduced features such as multiple flow tables, group tables, and advanced match fields, enhancing support for multi-tenant networks, traffic engineering, and dynamic load balancing [7]. These enhancements have strengthened OpenFlow's role in managing modern, high-performance SDN environments.

A. OpenFlow Components

An OpenFlow switch consists of several essential components that enable communication between the controller and the data plane, improving network efficiency and flexibility. Figure 1.1 illustrates its main

components, including the OpenFlow secure channel, OpenFlow ports, flow tables, a metric table, and a group table.

- **OpenFlow Secure Channel:** The OpenFlow secure channel establishes an encrypted communication link between the switch and the SDN controller, protecting control messages from unauthorized access and tampering. This ensures the integrity of operations such as dynamic flow table updates and performance monitoring, which are critical for maintaining network security.
- **OpenFlow Ports:** OpenFlow ports serve as interfaces that facilitate packet forwarding and external communication. They are categorized as follows:
  - **Physical Ports:** Hardware interfaces (e.g., Ethernet ports) that connect to physical network links.
  - **Logical Ports:** Software-defined interfaces representing virtual links, tunnel endpoints (e.g., GRE, VXLAN), or aggregated links.
  - **Reserved Ports:** Special-purpose ports supporting OpenFlow operations, such as directing unmatched packets to the controller (`CONTROLLER`) or replicating packets to all active ports (`FLOOD`).
- **Flow Tables:** Flow tables define packet-handling rules based on header fields such as IP addresses, port numbers, and protocol types.
  - **Matched packets** are processed according to predefined actions, such as forwarding, modifying headers, or dropping packets.
  - **Unmatched packets** are sent to the controller, which determines the appropriate action and updates the flow table dynamically.
- **Metric Table:** The metric table collects performance data, including packet counts, byte counts, and flow durations. This information helps the SDN controller monitor network health, identify congestion, and optimize traffic flow.
- **Group Table:** The group table extends flow table functionality by enabling advanced packet processing:
  - **Multicast and Broadcast:** Replicates packets to multiple ports.
  - **Load Balancing:** Distributes traffic across multiple paths to optimize network resources.
  - **Failover Mechanisms:** Redirects traffic to backup paths in case of link failures, enhancing network resilience.

These components enable OpenFlow switches to provide programmable, dynamic packet forwarding, ensuring efficient interaction with the SDN controller while supporting modern network requirements such as scalability and flexibility.

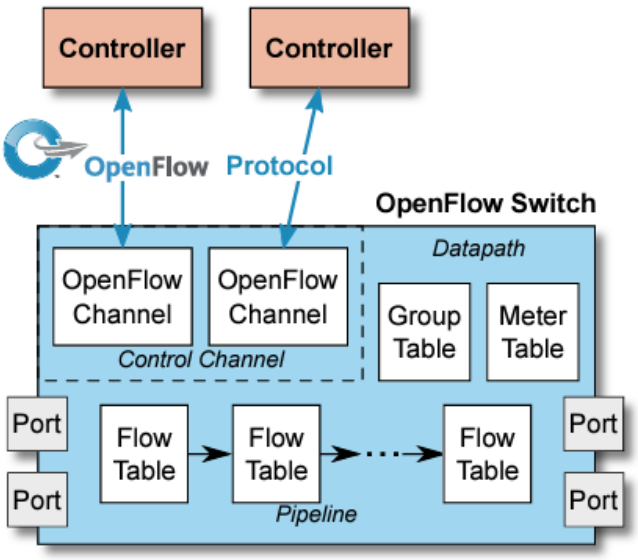


Figure 1.2: Main components of an OpenFlow switch

B. Flow Control Mechanism

As shown in Figure 1.3, each flow table entry consists of match fields, priority, counters, instructions, timeouts, a cookie, and flags. Match fields identify packets based on parameters such as ingress ports, Ethernet headers, IP addresses, and protocol-specific fields, including pipeline metadata. Priority determines rule evaluation order, ensuring higher-priority rules are processed first. Counters track the number of packets and bytes matching a rule, aiding in traffic monitoring and optimization. Instructions define actions for matched packets, such as forwarding, modifying headers, dropping, or additional processing. Timeouts specify how long a rule remains active: idle timeouts remove rules after inactivity, while hard timeouts enforce a fixed duration. The cookie uniquely identifies flow entries for tracking and updates. Flags provide control options, such as making rules persistent across switch reboots to enhance reliability.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

Figure 1.3: Main components of a flow entry in a flow table.

OpenFlow switches follow a match-action model, where incoming packets are evaluated against flow table rules. Upon receiving a packet, the switch performs an initial lookup in the first flow table and proceeds with lookups in subsequent tables if necessary to find a matching rule based on parameters such as ingress ports and IP addresses. If a match is found, the corresponding action is applied. If no rule matches, the packet is either dropped or forwarded to the SDN controller using an OpenFlow Packet-In message, as defined by the default rule. This behavior is illustrated in Figure 1.4.

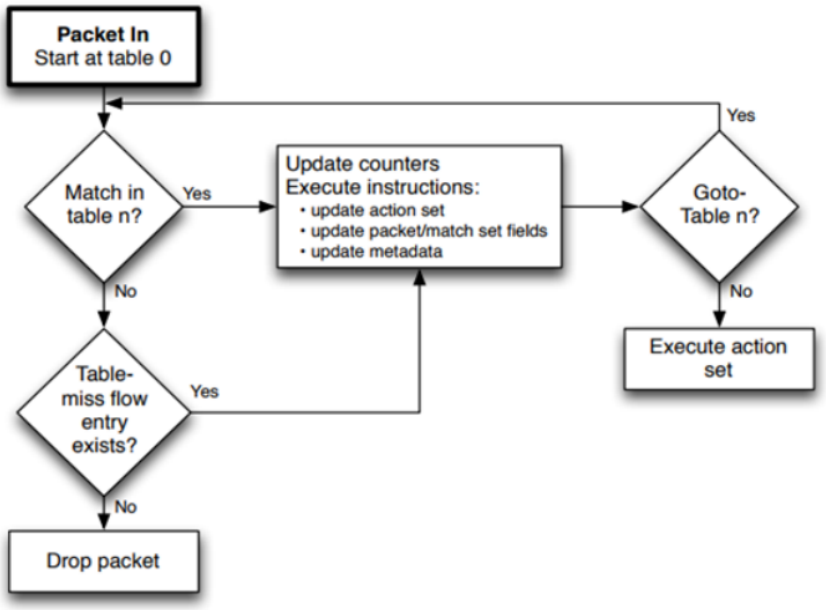


Figure 1.4: Packet processing in an OpenFlow switch

1.1.3 Threats in SDN Architecture

Security threats in SDN can be categorized into five groups, each affecting a specific layer or interface of the architecture: the application layer, the northbound interface, the control plane, the southbound interface, and the data plane. This classification helps identify vulnerabilities at each layer and interface, highlighting the specific threats they are exposed to.

A. Threats in Application Plane

The application layer in SDN provides a vendor-independent interface that enables unified access to network resources. This layer allows users to interact with and manage network components without reliance on specific vendor solutions. However, its openness introduces vulnerabilities that could be exploited by malicious software. While designed to enhance flexibility and security, this transparency may also create potential attack vectors. Table 1.1 presents key threats within the application layer.

Table 1.1: Summary of threats in the application layer

Threat	Description and Solutions
Fraudulent Rule Manipulation Attack	Malicious applications can create rule conflicts or overwrite existing flow rules, leading to unintended network behavior and performance disruptions. Such attacks can also weaken security measures, compromising network integrity and reliability. Additionally, these applications may generate excessive control messages, overwhelming the flow entry table and causing misdirection in network operations. Several mechanisms have been developed to detect and mitigate fraudulent rule manipulation attacks in SDN environments. These include the multi-control plane approach [8] and the state comparison technique [9].

Threat	Description and Solutions
Network Service Adversary Attack	Malicious applications can disrupt network services by executing arbitrary code to hijack the control plane or selectively drop packets, leading to network instability. Such attacks compromise system integrity and reliability, causing severe operational issues. Additionally, they may run infinite loops that overload the control plane, resulting in legitimate packet loss or rule flushing. To counter these threats, SDN environments employ behavior profiling and statistical anomaly detection methods [10] to detect malicious applications and prevent service disruptions and control plane hijacking.
Unauthorized Access Attack	Due to inadequate integrity and consistency controls, malicious applications may exploit their granted permissions to execute harmful activities, threatening network security and functionality. These unauthorized actions can compromise the integrity and trustworthiness of the system, creating serious vulnerabilities. To mitigate this risk, [11] and [12] propose solutions that introduce an access control layer to regulate application permissions and prevent unauthorized actions.

B. Threats at the Northbound Interface

The open access provided by the Northbound interface to application requests and control functions makes it a potential attack vector. Attackers can exploit these vulnerabilities to overwhelm system resources, interfere with service chaining, and impact network reliability. Table 1.2 summarizes the primary threats at the Northbound interface.

Table 1.2: Summary of threats at the northbound interface

Threat	Description and Solutions
Resource Exhaustion Attack	Malicious applications in SDN can overload the system with excessive requests or traffic, consuming bandwidth and straining the SDN controller's processing capacity. They may also discard valid packets, disrupting control messages between the controller and applications. This can delay or block legitimate processes, leading to degraded network performance and increased vulnerability. Mitigation measures include rate-limiting, traffic monitoring, and anomaly detection to maintain network stability.
Service Chain Jamming Attack	Attackers can interfere with service chaining by disrupting dependencies, leading to performance degradation, delays, or service failures. Ensuring that network applications operate accurately and reliably is essential. NICE [13] helps verify application correctness by analyzing and validating service interactions within SDN environments.

C. Threats in Control Plane

As the core decision-making entity in SDN, the control plane is a prime target for attackers seeking to disrupt network stability, manipulate traffic flows, or inject unauthorized modifications into network operations. Table 1.3 presents the key threats affecting the control plane and their corresponding mitigation strategies.

Table 1.3: Summary of threats in the control plane

Threat	Description and Solutions
Control Plane Hijacking Attack	An attack where an unauthorized entity gains control of the network, often through malicious applications. It exploits vulnerabilities in confidentiality and integrity, allowing attackers to inject and execute malicious commands. These actions can terminate critical applications, modify flow tables, and destabilize the control plane. Mitigation strategies include ControllerSEPA [14] and Byzantine fault-tolerant techniques [15].
Topology Poisoning Attack	This attack exploits authentication and integrity vulnerabilities to create fake network links and manipulate network topology. Attackers can alter the network structure, compromising its stability and security. TopoGuard [16] helps mitigate such threats but has limitations, particularly in detecting attacks during host migrations. Further research is needed to enhance protection against topology manipulation.
Denial of Service Attack	The centralized SDN control plane is vulnerable to DoS attacks that overwhelm it with excessive traffic, depleting critical resources and disrupting network operations. Mitigation methods include probabilistic proxying, blacklisting mechanisms [17], and collaborative intelligence systems [18] to manage traffic and prevent resource exhaustion.
Message Injection Attack	Weak integrity and insufficient authentication expose networks to message injection attacks, where incorrect headers disrupt network communication. Solutions like PacketChecker [19] and the INSPECTOR device [20] help detect and mitigate these threats. However, existing solutions require further improvements to ensure comprehensive protection.
Malicious Application Attack	The open programmability of the SDN control plane increases risks posed by malicious applications. These applications can execute harmful actions, such as modifying system time or disrupting critical transactions. Security measures, including NOSArmor [21] and permission-based detectors [22], aim to detect and prevent these threats. However, challenges remain in ensuring broader implementation and minimizing performance overhead.

D. Threats at the Southbound Interface

In SDN, securing the southbound interface between the control plane and the data plane is crucial for maintaining network stability and preventing unauthorized access. However, this interface is vulnerable to threats that can compromise availability, integrity, and confidentiality, leaving the network exposed to various attacks. Table 1.4 summarizes these key threats.

Table 1.4: Summary of threats at the southbound interface

Threat	Description and Solutions
Man-in-the-Middle Attack	An attacker intercepts communication between the control and data planes, often exploiting unencrypted transmissions. This can lead to credential theft and unauthorized access to control plane functions. A demonstration of this attack on the OpenDaylight controller used the Ettercap tool [23]. Mitigation strategies include Global Flow Table-based detection techniques to identify suspicious activities [24].

Threat	Description and Solutions
Eavesdropping or Sniffing Attack	Attackers intercept unencrypted communication between the control and data planes, exposing sensitive information such as network configurations and policies, which compromises confidentiality and security. Multipath solutions, such as those proposed by [25], mitigate these threats by dynamically switching paths and blocking acknowledgment responses from malicious sources.
Resource Exhaustion Attack	Attackers flood the Southbound Interface with excessive traffic, depleting critical resources such as controller processing capacity, switch flow table space, and available bandwidth. This can cause delays, packet drops, or outages, disrupting communication between the control and data planes and degrading network performance. FloodDefender [26] and FloodShield [27] help mitigate these attacks by monitoring traffic and dynamically managing resource allocation.

E. Threats in Data Plane

The data plane executes forwarding decisions based on rules installed by the control plane, making it vulnerable to security threats such as unauthorized access, denial of service, fraudulent rule insertion, and resource saturation. These threats can compromise network integrity, availability, and confidentiality. Table 1.5 provides an overview of these key threats.

Table 1.5: Summary of threats in the data plane

Threat	Description and Solutions
Unauthorized Access Attack	The lack of robust authentication and integrity verification increases the risk of unauthorized access attacks, where adversaries exploit end-hosts to carry out malicious activities. Without proper safeguards, attackers can manipulate network traffic or disrupt operations. Implementing strong authentication mechanisms that verify incoming flow requests against trusted credentials helps prevent unauthorized access [28].
Denial of Service Attack	Due to limited security measures in the data plane, SDN systems are vulnerable to Denial-of-Service (DoS) attacks. Attackers exploit memory and processing limitations to overwhelm the network and disrupt operations. Mitigation strategies include control plane clustering techniques using ASVM (Advanced Security Virtual Machine) [29] and monitoring algorithms [30].
Side Channel Attack	Timing gaps or flow configuration delays in the data plane may be exploited to infer sensitive network information. Such vulnerabilities allow attackers to deduce network activity, configurations, and system behavior. In SDN environments, where the control plane manages traffic, these attacks threaten security and confidentiality. Fingerprinting techniques help detect and mitigate such threats [31, 32].
Fraudulent Rule Insertion Attack	Weak integrity and authentication mechanisms allow attackers to inject fraudulent flow rules, leading to network poisoning and unexpected behavior that disrupts operations. These attacks compromise SDN reliability and security, emphasizing the need for robust validation mechanisms to detect and prevent unauthorized rule modifications.

Threat	Description and Solutions
Resource Saturation Attack	Limited processing power and weak authentication mechanisms in the data plane make it vulnerable to saturation attacks. Attackers flood the network with excessive traffic or malicious flow requests, degrading performance and causing service disruptions. Effective flow management techniques and traffic validation [33] help mitigate these threats, while anomaly detection [34] enhances attack detection.

1.2 Traditional Security Methods in SDN

This section examines traditional network security mechanisms and their adaptation to SDN. While SDN introduces programmability and centralized control, established security approaches such as network segmentation, Virtual LANs (VLANs), access control lists (ACLs), firewalls, and intrusion detection systems remain widely implemented. These methods enhance security by isolating traffic, enforcing access control, and detecting threats. In SDN, their effectiveness is further improved through automation and centralized management.

1.2.1 Network Segmentation

Network segmentation is a long-standing security practice that extends to SDN and other modern network architectures. It involves dividing a network into smaller segments to simplify management, enhance security, and restrict access to sensitive resources. This approach helps mitigate cybersecurity incidents by preventing unauthorized access and limiting the spread of threats during breaches. In traditional networks, segmentation relies on routing tables and carefully planned IP addressing to avoid conflicts. With SDN, segmentation is implemented at the software level, providing greater flexibility and efficiency while enabling seamless integration of firewall rules and security policies during segment creation.

SDN's centralized management allows administrators to enforce security policies consistently across all network segments from a single controller. This programmability enables organizations to tailor segmentation and security configurations to their specific needs. However, managing multiple network segments in dynamic SDN environments presents challenges, particularly in maintaining consistent policy enforcement. Centralized policy management, combined with robust monitoring tools, is crucial for visibility and compliance. Effective monitoring solutions provide a comprehensive network-wide view, ensuring uniform security across all segments.

1.2.2 Virtual LANs

Virtual LANs play a crucial role in enhancing security and network efficiency within SDN. They segment networks into multiple isolated broadcast domains, reducing congestion and improving access control by ensuring traffic reaches only its intended destinations. This segmentation limits the impact of security breaches, containing potential threats within specific VLANs rather than exposing the entire network.

Despite their benefits, VLANs are susceptible to security risks, with unauthorized access being a primary concern. Attack techniques such as VLAN hopping allow attackers to bypass VLAN boundaries, while MAC spoofing enables them to impersonate legitimate devices and gain unauthorized access. Additionally, misconfigurations and weak access controls may lead to unintended communication between VLANs, increasing security vulnerabilities.

To mitigate these risks, organizations must adopt VLAN security best practices, including VLAN tagging for proper isolation, port security to restrict unauthorized device access, and access controls to regulate inter-VLAN communication. Regular audits and continuous monitoring help identify and address configuration weaknesses, ensuring a more secure network environment.

In SDN architectures, VLAN security is further enhanced through programmable features that enable dynamic VLAN management, automated threat detection, and real-time security policy updates. This flexibility strengthens security while simplifying VLAN configuration and maintenance.

1.2.3 Access Control Lists

Access Control Lists are a traditional and effective method for managing user permissions and regulating access to network resources across various computing platforms. In security applications, ACLs enforce access controls through a set of predefined rules and operate on a ”first match” basis, meaning traffic is either allowed or blocked based on the first applicable rule. Once a match is found, no further rules are evaluated, ensuring efficient decision-making. Furthermore, ACLs enhance reliability by enforcing access control policies for databases and web servers, allowing only authorized users to access resources while blocking unauthorized ones.

ACLs are widely used in SDN to enforce security policies by restricting unauthorized access and controlling network traffic. While they effectively filter out malicious or unwanted traffic, misconfigurations may inadvertently restrict legitimate requests, disrupting normal operations. Additionally, implementing ACLs in large networks can be complex due to the need to define and maintain numerous access rules. Manual configuration increases the risk of errors, such as mistakenly denying access to authorized users or unintentionally granting access to unauthorized ones, leading to security vulnerabilities or operational disruptions.

Proper ACL deployment requires a thorough understanding of network architecture, traffic patterns, and access requirements to ensure effective security enforcement. However, their reliance on predefined rules introduces certain limitations. ACLs may not account for all attack vectors or adapt to emerging threats, which can limit their ability to enforce fine-grained security policies. As a result, network switches may struggle to apply security measures at the per-flow or per-session level, reducing their overall effectiveness in protecting the network.

Despite these challenges, ACLs remain an essential component of a defense-in-depth strategy, providing an additional layer of security by regulating access and protecting network resources. When properly managed, they help organizations enforce security policies while minimizing the risk of unauthorized access and data breaches.

To address some of ACLs' inherent limitations, SDN-based ACL deployment offers a centralized

approach through an SDN controller, which dynamically manages and distributes ACL rules across all network devices. Instead of configuring rules on individual routers or switches, administrators define access policies in the controller, which then pushes updates to all connected devices. This method ensures uniform policy enforcement, reduces administrative overhead, and enhances scalability. Furthermore, SDN's centralized architecture enables seamless integration with management protocols such as NETCONF and REST APIs, allowing for automated ACL configuration, policy enforcement, and real-time security adjustments based on evolving threats.

1.2.4 Firewalls and Intrusion Detection Systems

Firewalls serve as the first line of defense in network security, regulating traffic based on predefined rules to maintain a controlled data flow. When integrated with SDN, firewalls provide scalable and dynamic security, as their policy enforcement mechanisms are programmable and adaptable to evolving threats.

Intrusion Detection Systems complement firewalls by monitoring network traffic for suspicious activity and generating real-time alerts when potential threats are detected. IDS are categorized into two types, each employing distinct techniques to identify threats: Signature-Based IDS and Anomaly-Based IDS.

Signature-Based IDS identify threats using predefined attack signatures based on documented malicious activity. They continuously monitor network traffic, comparing incoming data against a stored database to detect known threats with low false positive rates. However, they struggle to detect new or evolving threats that lack existing signatures, such as zero-day attacks. Attackers can modify malware structures through obfuscation or polymorphism, making detection difficult. To address these limitations, machine learning techniques are increasingly used to develop adaptive signatures that identify variations of existing threats without requiring frequent manual updates.

Anomaly-Based IDS detect previously unknown threats by identifying deviations from normal network behavior. Using machine learning, they establish a baseline of expected network activity and flag deviations for further inspection, helping detect zero-day attacks that signature-based systems might miss. These systems continuously refine their models with real-time data, allowing them to adapt to evolving cyber threats. While more effective at identifying novel attack behaviors, anomaly-based IDS may have a higher false positive rate if improperly trained.

In SDN environments, IDS leverage centralized architectures and tools such as OpenFlow messages (*"StatsRequest"* and *"StatsResponse"*) to gather detailed traffic statistics, enhancing anomaly detection and response capabilities. Machine learning further improves detection accuracy and accelerates response times, enabling faster threat mitigation.

1.3 Intrusion Detection System Using Machine Learning Techniques

The following discussion outlines key stages of the machine learning workflow, detailing model training, evaluation, and application to new data. It also explores essential techniques that enhance security in SDN by detecting and mitigating cyber threats through various learning approaches. Additionally, it

presents widely used intrusion detection datasets that serve as benchmarks for training and evaluating machine learning models in network security.

1.3.1 Machine Learning Workflow

A machine learning process typically consists of two primary stages: training and prediction. During training, algorithms analyze a dataset to identify patterns, correlations, and underlying structures, enabling the development of a model that captures critical information. Once trained, the model is used to forecast or categorize new, previously unseen data. This structured approach allows machine learning systems to enhance decision-making independently, making them effective for tasks such as intrusion detection, fraud prevention, and image analysis.

A key strength of machine learning models is their ability to refine their understanding of patterns as they process more data, improving accuracy with each iteration or update. This adaptability is particularly important in dynamic environments that require swift responses to emerging patterns or threats. By continuously learning from real-time data, models remain effective and produce relevant outputs over time.

A crucial component of the machine learning pipeline is data preparation, which ensures that data is structured and optimized for modeling. This stage involves data cleaning to remove noise, handling missing values, and formatting data correctly. Proper feature selection also plays a key role in identifying the most influential variables, enhancing model accuracy and efficiency.

After preprocessing, the data is typically divided into training and testing sets. The training set is used to build and fine-tune the model, while the testing set evaluates its performance. A well-performing model on the testing set indicates strong generalization to new data, which is essential for real-world applications.

Figure 1.5 illustrates the machine learning workflow, covering data preprocessing, training, and prediction stages. As data evolves, models adapt by learning new patterns, ensuring consistent performance improvements and maintaining reliability over time.

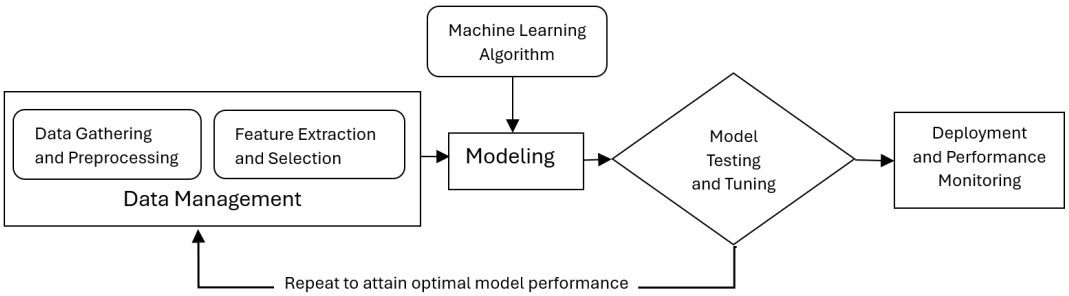


Figure 1.5: Machine learning workflow

1.3.2 Machine Learning Techniques and Approaches for Detecting Attacks

Machine learning enhances Software-Defined Networking security by enabling proactive detection of evolving threats. Figure 1.6 presents key machine learning techniques used for identifying vulnerabilities and mitigating attacks.

Supervised learning utilizes labeled datasets, allowing models to learn established input-output relationships. This approach is particularly effective in identifying known attack types by recognizing distinct patterns associated with malicious activities.

Unsupervised learning identifies anomalies without predefined labels, making it valuable for zero-day attack detection. Clustering methods, such as K-Means and DBSCAN, help detect deviations from normal traffic patterns in SDN environments.

Reinforcement learning improves security decision-making by dynamically adjusting policies based on network feedback. This allows SDN controllers to optimize attack mitigation strategies in real time.

Semi-supervised learning leverages both labeled and unlabeled data to enhance detection when labeled datasets are scarce. This is useful in SDN security, where collecting comprehensive labeled traffic data is challenging.

Deep learning algorithms utilize neural networks with multiple layers to detect attacks. These approaches provide advanced feature extraction and classification capabilities. Common deep learning models for SDN security include Conventional Neural Networks, Recurrent Neural Networks, and Long Short-Term Memory networks.

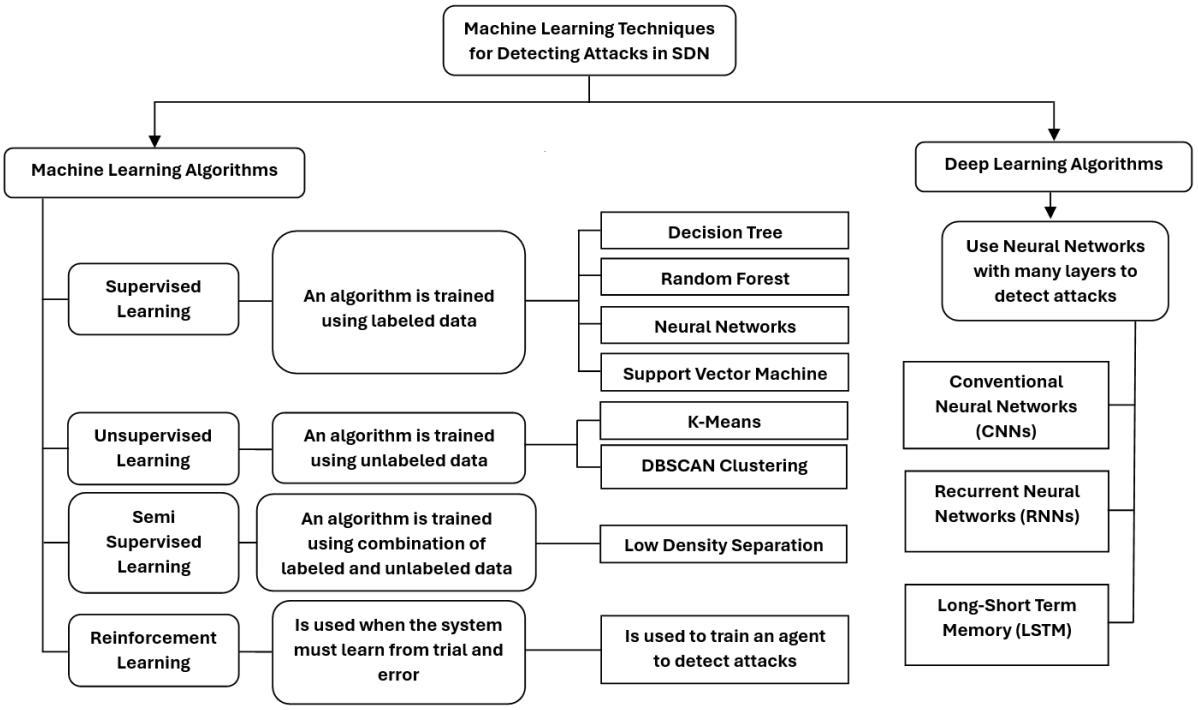


Figure 1.6: Commonly used machine learning techniques to detect attacks in SDN

Table 1.6 summarizes machine learning algorithms commonly applied in Network Intrusion Detection Systems (NIDS), highlighting their strengths and limitations.

Table 1.6: Overview of commonly used machine learning algorithms in network intrusion detection systems

Algorithm	**Description**	**Benefits**	**Drawbacks**
Decision Tree (DT)	Splits data into subsets based on feature values, forming a tree structure for classification and regression.	Simple, interpretable, efficient.	Prone to overfitting, sensitive to data changes.
Random Forest (RF)	An ensemble of decision trees trained on different data subsets to improve accuracy and reduce overfitting.	Robust, accurate, reduces overfitting.	Requires more memory, less interpretable.
Neural Networks (NNs)	Connects layers of neurons to learn patterns from data, useful for various tasks like image and language processing.	Captures complex patterns, highly adaptable.	Computationally expensive, prone to overfitting.
Support Vector Machine (SVM)	Finds the optimal hyperplane to separate data classes, using kernel functions for nonlinear classification.	Effective in high-dimensional spaces, robust to outliers.	Computationally expensive, requires careful tuning.
K-Means Clustering	Groups data into k clusters based on similarity. Used for segmentation tasks.	Simple, fast, scalable.	Sensitive to initial centroid placement, requires predefined k.
DBSCAN Clustering	Identifies clusters based on data density, handling arbitrary cluster shapes.	Detects non-spherical clusters, robust to outliers.	Computationally demanding, sensitive to distance thresholds.
Convolutional Neural Networks (CNNs)	Extracts spatial patterns from data, commonly used in computer vision tasks such as image recognition and segmentation.	Highly effective in feature extraction.	High computational cost, requires large labeled datasets.
Recurrent Neural Networks (RNNs)	Processes sequential data while maintaining temporal dependencies.	Captures time-based patterns effectively.	Prone to vanishing gradients, difficult to train on long sequences.
Long Short-Term Memory (LSTM)	A variant of RNNs that better handles long-term dependencies.	Mitigates vanishing gradient issues.	Computationally expensive.

In SDN, several machine learning approaches are employed to enhance attack detection and mitigation by analyzing network patterns and identifying anomalies. Table 1.7 summarizes various ML approaches used to secure SDN networks.

Table 1.7: Machine learning approaches to detect and mitigate attacks in SDN

ML Approach	**Description**	**Related Papers**
Behavioral Profiling	Uses ML to model normal network behavior and detect deviations indicating potential attacks.	[35]
Feature Extraction and Classification	Extracts key network attributes to classify traffic as benign or malicious.	[36]

Continued on next page

	Description	Related Papers
Adaptive Models and Real-Time Analysis	Dynamically adjusts detection strategies and performs real-time threat analysis.	[37]
Predictive Analytics	Uses historical data to predict and prevent future threats.	[38], [39]
Ensemble Methods	Combines multiple ML models to enhance detection accuracy.	[40], [41]
Unsupervised Learning for Zero-Day Attacks	Detects unknown attack patterns using clustering and anomaly detection.	[42]

1.3.3 Datasets for Intrusion Detection

Table 1.8 provides an overview of widely used datasets for intrusion detection, helping researchers train and evaluate machine learning models. The effectiveness of intrusion detection systems depends on dataset diversity, as rich and varied data improve pattern recognition and intrusion detection accuracy. Datasets containing both normal and malicious traffic are particularly valuable for ensuring model reliability across different network environments.

Creating such datasets requires extensive data collection, annotation, and balancing to reflect real-world network behavior. Well-established datasets serve as standardized benchmarks, enabling consistent evaluations and fostering advancements in IDS technologies.

Table 1.8: Commonly used machine learning datasets for intrusion detection in the literature

Dataset	Description	Date and References
Real-world datasets	Collected from live operational networks, capturing authentic traffic, system logs, and security incidents. Unlike synthetic datasets, they provide real-world variability, aiding in intrusion detection research.	Not applicable
Synthetic datasets	Artificially generated to simulate specific attack scenarios, traffic anomalies, or rare events, enabling controlled testing of intrusion detection models.	Not applicable
DARPA	Developed by MIT Lincoln Laboratory, this dataset contains simulated network communication, including normal and attack traffic for intrusion detection system evaluation.	1998 [43]
KDD Cup 1999	Derived from DARPA, this dataset simulates military network traffic, featuring normal operations and multiple attack categories such as probing and DDoS.	1999 [44]
CAIDA	Offers various datasets for cybersecurity research, including real-world traffic data useful for studying DDoS attacks and routing anomalies.	2005 [45], [46]
NSL-KDD	An improved version of KDD Cup 1999, addressing redundancy and imbalance issues to provide a more realistic dataset for intrusion detection research.	2009 [47]
CTU-13 Dataset	Contains 13 botnet scenarios collected from real-world network traffic, widely used for botnet detection and network security analysis.	2011 [48]

Continued on next page

Dataset	Description	Date and References
UNSW-NB15	Created by the Cyber Range Lab at UNSW, this dataset includes real-world network traffic with nine attack types, making it a benchmark for intrusion detection.	2015 [49]
BGPStream	A real-time and historical dataset for Border Gateway Protocol (BGP) data, supporting research in routing security and anomaly detection.	2015 [50]
CIC-IDS 2017	Captures five days of network activity, featuring over 80 attributes of both normal and attack traffic, including botnets, brute force, and infiltration.	2017 [51], [52]
CSE-CIC-IDS2018	Includes various cyber threats such as brute force attacks, DDoS, and web-based exploits, offering detailed network behavior profiling.	2018 [53]
CIC-DDoS2019	Contains 46 million records of modern DDoS attack types, providing network traffic in PCAP and CSV formats for extensive analysis.	2019 [54], [55]
InSDN	Designed for SDN intrusion detection, featuring 361,000 traffic instances covering multiple attack types, including botnets and web-based threats.	2020 [56], [57]

1.4 Optimizing ML-Based IDS Scalability in SDN

The rapid growth of data generated by interconnected devices presents significant scalability challenges for ML-based IDSs in large-scale SDN environments. These systems must efficiently handle and analyze vast amounts of traffic while maintaining high performance, accuracy, and real-time threat detection capabilities. As network traffic increases, delays in processing may impact real-time analysis, raising the risk of undetected security threats. To overcome these challenges, it is essential to implement strategies that improve data processing efficiency, optimize resource allocation, and enhance real-time detection. The following discussion explores key architectural approaches and methods to address these issues.

1.4.1 Scalable Architectures for Large-Scale Deployments

Handling increasing traffic loads while maintaining system efficiency and responsiveness is a critical challenge for IDSs. Scalable architectures address this issue through the following approaches:

- **Distributed Systems:** Technologies such as Apache Kafka and Hadoop enable IDSs to distribute workloads across multiple nodes, facilitating parallel data processing. This ensures high performance even during peak traffic periods. In SDN-based IoT environments, processing data closer to the source at edge or fog layers reduces reliance on centralized servers and minimizes latency. A hybrid approach that integrates edge, fog, and centralized systems enhances local resource utilization while leveraging centralized computing power for real-time intrusion detection and efficient large-scale traffic management.

- **Cloud-Based Deployments:** Cloud infrastructures provide flexible resource allocation, dynamically adjusting computational capacity based on traffic demands. This elasticity ensures seamless scaling during traffic surges, maintaining performance and responsiveness while handling fluctuating network loads.

- **Modular Design:** A modular IDS architecture divides the system into independent components, such as feature extraction, classification, and alerting. This allows each module to scale separately based on demand, optimizing resource allocation and simplifying updates without affecting the entire system.

1.4.2 Balancing Accuracy and Resource Utilization

Maintaining high accuracy while optimizing computational efficiency is critical for improving IDS scalability and performance. This balance can be achieved through various techniques that enhance processing efficiency without compromising threat detection reliability. The following methods contribute to this objective:

- **Data Reduction via Sampling:** This approach reduces the data volume by selecting a subset that accurately represents the entire dataset. By preserving essential information while lowering overall data size, sampling accelerates both training and evaluation, making real-time processing more efficient [58].

- **Epsilon-Dominating Pattern Mining:** This method identifies frequently occurring patterns or features in a dataset and clusters similar instances, effectively reducing dataset size. By focusing on dominant structures, it enhances analytical efficiency in threat detection while preserving the quality of insights [59].

- **Feature Selection:** A fundamental technique in intrusion detection, feature selection identifies the most relevant attributes for detecting malicious activities [60]. This approach enhances classification accuracy, reduces computational overhead, and improves overall model efficiency. Feature selection methods include Filter [61], Wrapper [62], and Embedded [63] techniques. Table 1.9 provides a comparison of these approaches.

Table 1.9: Feature selection methods: Filter, Wrapper, and Embedded

Technique	Description	Key Characteristics
Filter	Select features based on their individual properties, without considering model performance.	Simple, computationally efficient, but may include irrelevant features. Examples: Univariate feature selection, correlation, chi-squared, mutual information.
Wrapper	Evaluate subsets of features by using a specific predictive model, taking interactions between features into account.	Can provide superior feature subsets, but computationally expensive. Examples: Forward selection, backward elimination, Recursive Feature Elimination (RFE).
Embedded	Feature selection is integrated into the model training process, using model-specific criteria for selecting features.	Combines advantages of filter and wrapper methods, reducing overfitting and computational cost. Examples: Lasso regression, Ridge regression, Decision Trees, Neural Networks.

- **Dimensionality Reduction Techniques:** These methods reduce the number of dimensions in a dataset, streamlining data management while preserving key characteristics. This process improves model efficiency and performance [64]. Common approaches include Principal Component Analysis (PCA) [65], Linear Discriminant Analysis (LDA) [66], and t-SNE (t-distributed Stochastic Neighbor Embedding) [67]. Table 1.10 provides a comparison of these techniques.

Table 1.10: Summary of PCA, LDA, and t-SNE

Aspect	PCA	LDA	t-SNE
Type	Unsupervised, linear	Supervised, linear	Unsupervised, non-linear
Objective	Maximize variance, reduce dimensionality	Maximize class separability	Preserve local structure for visualization
Core Method	Eigen-decomposition of covariance matrix or SVD (Singular Value Decomposition)	Maximize between-class variance, minimize within-class variance	Gradient descent to optimize probability distributions
Applications	Preprocessing, visualization, noise elimination	Feature selection, classification, face recognition	Visualization of high-dimensional clusters and distributions
Advantages	Retains maximum variance, efficient, interpretable	Enhances class separation, improves classification performance	Reveals complex structures, strong visualization capabilities
Limitations	Sensitive to scaling, assumes linearity	Assumes normal distribution and equal covariance matrices	Sensitive to parameter selection, computationally intensive for large datasets

1.5 Adversarial Threats in Machine Learning Models

Adversarial threats pose a major challenge to machine learning models, especially in SDN environments. Attackers generate specially crafted inputs to mislead or exploit ML models, which can undermine system security, reliability, and decision-making accuracy. Given that ML is integral to monitoring and controlling SDN networks, adversarial attacks can severely impact network performance, stability, and security [68].

1.5.1 Types of Adversarial Attacks

Adversarial attacks on machine learning models fall into three main categories: evasion attacks, poisoning attacks, and model inversion attacks [69]. Each of these exploits different weaknesses in ML systems and introduces specific risks in SDN security.

- **Evasion Attacks**: In these attacks, adversaries alter input data to deceive ML models into making incorrect classifications or predictions. This method is commonly used to bypass security mechanisms such as network intrusion detection systems, spam filters, and authentication protocols. By modifying key characteristics of input data, attackers can evade detection and weaken the model's reliability in real-time applications.

- **Poisoning Attacks**: Poisoning attacks compromise the training phase of machine learning models by introducing manipulated or misleading data into the training set. This tampering distorts the learning process, causing the model to learn incorrect patterns, leading to misclassifications during both training and inference. Such attacks can degrade model performance, weaken generalization capabilities, and create security vulnerabilities in SDN environments.

- **Model Inversion Attacks**: These attacks aim to extract sensitive information from a trained model by analyzing its internal structure and learned features. Adversaries can use this analysis to infer private details from the training data, potentially compromising confidential or proprietary information. In SDN, where data privacy and security are crucial, model inversion attacks present a serious risk to network integrity and data protection.

1.5.2 Defense Techniques Against Adversarial Threats

To mitigate adversarial threats and strengthen the security of machine learning models in SDN, several techniques have been introduced. Among the most effective methods are adversarial training and feature space transformation [70]. Table 1.11 presents a comparison of these approaches in enhancing the robustness of machine learning models against adversarial attacks.

- **Adversarial Training**: This approach enhances model resilience by incorporating adversarially generated examples into the training process. By exposing the model to such perturbed inputs, it learns to recognize and counter adversarial manipulations, leading to improved generalization and more stable decision boundaries. However, adversarial training is computationally intensive and may be ineffective if adversarial examples are not well-designed.

- **Feature Space Transformation**: This method modifies the feature representation of data to make it less susceptible to adversarial alterations. Techniques such as PCA and LDA reduce dimensionality, making it harder for attackers to manipulate important features. Additionally, feature encoding can mask critical attributes, limiting the ability of adversaries to exploit them. While feature space transformation enhances model security, it also introduces challenges such as higher computational overhead, reduced interpretability, and potential accuracy loss. Changing feature representations or applying encoding techniques may remove essential information, which can affect classification performance in real-world applications.

Table 1.11: Comparison of adversarial training and feature space transformation for security enhancement

Technique	Description	Strengths	Challenges
Adversarial Training	Integrates adversarial examples into training data to enhance model robustness.	Improves generalization and decision-making.	Computationally intensive; error-prone.

Continued on next page

Table 1.11 – *Continued from previous page*

Technique	Description	Strengths	Challenges
Feature Space Transformation	Alters the feature space to obscure exploitable data characteristics using PCA, LDA, or feature encoding.	Increases resistance to manipulation.	Can reduce interpretability; accuracy trade-offs; significant computational overhead.

1.6 Deployment Challenges and Solutions in ML-Based IDS

Deploying ML-based IDSs in SDN environments requires a structured approach to ensure efficiency and compatibility. This section explores key deployment challenges, highlighting the importance of seamless integration with SDN controllers, interoperability with various network protocols, and optimized deployment strategies for effective operation.

1.6.1 Compatibility with Existing SDN Controllers

Integrating ML-based IDSs, particularly deep learning models, into SDN environments requires careful planning to ensure efficient operation. A key goal is to achieve real-time threat detection while accurately identifying and classifying malicious activities. The effectiveness of this process depends on how well machine learning models interact with SDN controllers. However, many SDN controllers are designed primarily for network management and lack built-in support for machine learning. This requires additional effort to establish communication between the SDN controller and external ML models, manage data exchange efficiently, and maintain real-time performance, making integration more complex.

Moreover, deploying IDSs in real-world SDN environments presents several challenges. The limited processing capacity of SDN controllers can make real-time flow monitoring and classification difficult, impacting both scalability and detection accuracy [71]. These challenges become more pronounced in single-controller architectures [72, 73, 74], where flow data must be continuously processed for real-time analysis. This constant workload places significant strain on memory and processing resources, potentially creating bottlenecks that limit scalability and degrade real-time detection performance.

To overcome these issues, improvements in SDN controller architecture [75] are necessary to enhance processing power, optimize memory usage, and improve compatibility with machine learning models. Additionally, adopting better resource management techniques and shifting to hybrid or distributed architectures can help distribute computational workloads across multiple controllers. These enhancements allow for better scalability and more effective deployment of ML-based IDSs in SDN environments.

1.6.2 Interoperability with Network Protocols

Ensuring interoperability in ML-based IDSs within SDN environments depends on their ability to process data from various network protocols, such as OpenFlow, NetFlow, and sFlow. OpenFlow provides detailed flow control capabilities by enabling OpenFlow-enabled switches to collect essential flow statis-

tics, including packet counts, byte counts, and flow durations. The SDN controller retrieves these statistics by sending FlowStatsRequest messages to switches, which respond with FlowStatsReply messages containing the requested data. These flow metrics play a crucial role in monitoring network performance and ensuring security.

In contrast, NetFlow aggregates flow-level metadata by grouping packets based on shared attributes such as IP addresses, port numbers, and flow durations. Meanwhile, sFlow periodically samples individual packets, capturing detailed header and payload information. While NetFlow offers a broader view of network traffic, sFlow provides more granular insights, making them complementary for comprehensive traffic analysis. To effectively utilize these data sources, ML-based IDSs must preprocess and standardize protocol-specific data into features suitable for machine learning models.

To safeguard data transmission to the ML-based IDS, Transport Layer Security (TLS) should be employed to maintain data integrity and confidentiality as it moves across the network. Once received, the IDS securely processes the information and communicates its findings to the SDN controller through standardized APIs and structured data models. This enables swift actions such as modifying flow rules or isolating harmful traffic, thereby enhancing the system's overall reliability and performance.

Maintaining ML-based IDSs operational during protocol updates is essential for ensuring consistent performance. Protocols like OpenFlow and NetFlow frequently undergo revisions that impact data structuring and transmission. To address these changes, protocol-agnostic encoding methods are applied to generalize protocol-specific details, ensuring compatibility across different protocol versions. Additionally, protocol-aware analysis refines detection methods to suit the characteristics of each protocol. For instance, an IDS that analyzes OpenFlow traffic may focus on identifying anomalies in flow rule changes, whereas a NetFlow-based IDS might monitor unusual variations in traffic volume. By tailoring detection techniques to protocol behavior, the system improves its accuracy in identifying security threats in evolving network environments.

1.6.3 Proposed Deployment Strategies

Deploying ML-based IDSs in real-world environments requires strategic planning to ensure both effectiveness and scalability. Early integration can improve performance, but developing reliable deployment models remains a challenge. Practical strategies are essential for meeting operational needs, maintaining system reliability, and ensuring a smooth transition to ML-based detection. Hybrid approaches and incremental deployment are effective solutions that help address key challenges and simplify integration into existing infrastructures.

A. Hybrid Architectures

A hybrid model combines conventional detection techniques with ML-based methods to enhance intrusion detection [76]. Traditional rule-based security systems, including NIDSs and HIDSs, are integrated with ML-driven anomaly detection to strengthen overall security. The following outlines three architectures that incorporate both signature-based and anomaly-based detection methods:

- **Collaborative Architecture**: In an SDN environment, hybrid IDS solutions can integrate signature-based and anomaly-based detection within a collaborative framework. Each detection mechanism operates independently, optimized for its specific approach. Signature-based IDS focuses on identifying predefined attack signatures, while anomaly-based IDS detects deviations from expected network behavior. Combining both methods improves the system's capability to detect both known threats and previously unseen attacks.

- **Parallel Architecture**: In SDN-based hybrid detection systems, a parallel architecture processes network traffic concurrently using both signature-based and anomaly-based IDS techniques. Both detection methods run at the same time, which helps generate alerts quickly and reduces delays in detection. The traffic is split across multiple processing units, enabling faster analysis and better scalability. By functioning in parallel, the system can manage larger datasets, identify a wider range of threats, and respond more quickly.

- **Cascade Architecture**: A cascade architecture organizes detection models into a series of decision layers, each addressing a specific task, beginning with simpler methods and progressing to more complex ones. The early stages use signature-based techniques to rapidly identify known threats and eliminate benign traffic. As the data advances through further stages, anomaly-based methods are applied to detect more complex or unfamiliar attacks. This tiered structure maximizes computational efficiency by processing simpler traffic first, minimizing unnecessary analysis in later stages. By filtering out irrelevant data early, the system allocates computational resources to more suspicious traffic, improving both efficiency and detection precision.

B. Incremental Deployment Approaches

A stepwise deployment strategy offers a structured approach for integrating ML-based IDSs into SDN infrastructures, reducing the risks of immediate full-scale implementation by enabling controlled integration of machine learning methods. Starting with limited deployments, such as processing selected traffic segments or analyzing specific data subsets, allows organizations to refine IDS performance and address issues like false positives before full-scale rollout. In the initial phase, ML models are trained and tested in real-time operations, ensuring continuous performance monitoring without disrupting network stability. Feedback from real-world network interactions fine-tunes IDS configurations, allowing early identification of weaknesses and gradual optimization. This incremental approach enhances threat detection while minimizing network disruptions and false alarms, enabling ML-based IDSs to adapt seamlessly to SDN environments, integrate smoothly with existing systems, and maintain overall network performance and security resilience.

1.7 Literature Review

Several approaches for dealing with various types of cyberattacks have been developed in the field of Software-Defined Networking. This section covers related studies organized by the types of attacks they

try to mitigate, such as DDoS, Man-in-the-Middle (MiTM), Low Rate DDoS (LR-DDoS), Application Layer Flooding, and others.

1.7.1 DDOS Attacks

Several methodologies have been developed for detecting and mitigating DDoS threats in SDN systems. One approach focuses on detecting flooding DDoS attacks on the SDN control plane by evaluating and analyzing the number of Packet-In requests sent to the SDN controller over a certain time period. Simulation findings show that this method exceeds 99% accuracy, with Bagging Tree (BT) being the most successful supervised learning strategy, reaching an accuracy of 99.64% [77]. Another approach combines information entropy and deep learning for DDoS detection, using an entropy-based method to identify the switch responsible for suspicious traffic entering the network and a deep learning module to classify packets. This technique outperforms standard algorithms such as SVM, DNN, and DT, with accuracy increases ranging from 4.25% to 8.20% [78]. The authors in [79] use machine learning to detect and mitigate botnets in SDN. This method mimics a botnet with Mininet and integrates a Python component with the RYU controller to successfully detect attack sources and apply flow rules for mitigating ongoing DDoS attacks. A comparative analysis of four machine learning algorithms for DDoS detection evaluated the effect of feature selection, showing that K-Nearest Neighbors (KNN), when trained with selected features, achieved an accuracy of 98.3%, exceeding the accuracy obtained with the complete feature set [80].

1.7.2 LR-DDOS Attacks

Several solutions have been developed to address LR-DDoS attacks. A hybrid model integrating CNN and LSTM networks [81] was proposed for detecting LR-DDoS threats, utilizing a small SDN network topology for training and testing sets. This model demonstrated a remarkable 99.998% accuracy, surpassing traditional models such as 1-Class SVM and Multi-Layer Perceptron (MLP). Another technique [82] utilizes dynamic flow rule deletion to prevent flow table saturation. It integrates a multi-feature detection system based on Factorization Machines (FM), and its performance is evaluated using the Area Under the Curve (AUC) metric. In addition, [83] presents a model-centric Weighted Federated Learning (WFL) approach for identifying LR-DDoS attacks in IoT networks controlled by SDN. This WFL model achieves an accuracy of 98.85%, sensitivity of 98.13%, F1-Score of 94.21%, and a low misclassification rate of 1.15%. Finally, a flexible architecture for detecting and mitigating LR-DDoS attacks incorporates an Intrusion Prevention System (IPS) on the controller and an externally placed Intrusion Detection System, facilitating efficient communication and flow rule creation upon identifying malicious traffic [84].

1.7.3 Other Attacks and Anomalies

The proposed approach in [85] employs a Genetic Algorithm alongside an SVM classifier to detect Slow HTTP DoS attacks. The dataset is prepared using a GNS3 SDN simulation and NetFlow records for analysis. The Genetic Algorithm identifies critical parameters, optimizing SVM performance. With

optimized parameters, the SVM classifier achieves an accuracy of 99.89%, an AUC of 99.89%, a true positive rate (TPR) of 99.5%, a false positive rate (FPR) of 0.18%, and a false negative rate (FNR) of 0.05%.

For detecting DNS flood attacks, the authors in [86] propose a hybrid deep learning model that combines CNN and LSTM cells. Tested on the CICIDS dataset, this model achieves 99.87% accuracy, surpassing traditional classifiers such as SVM and Artificial Neural Network (ANN) without requiring normalization or feature removal. The proposed architecture consists of three stages: network traffic measurement, DNS flood analysis, and anomalous traffic classification. Similarly, [87] employ various machine learning techniques on the SDN controller for detecting DNS amplification attacks. A comparative analysis indicates that decision trees reduce detection time, while SVM and Gradient Boosting Classifier (GBC) achieve higher accuracy.

The authors in [88] propose a method for detecting TCP-SYN flood and ICMP flood attacks using entropy and logarithmic calculations. This method employs KNN to identify the K nearest points based on the Euclidean distance to the current entropy or logarithm value. When K is set to 9, evaluation on both the CAIDA 2007 dataset and simulated traffic shows an accuracy of over 99%.

In [89], the authors created a dataset containing only TCP traffic from a real-world network. They used two key attributes, the number of connected devices per second and peak/off-peak time indicators, to train three supervised learning classifiers: SVM, neural network, and Naïve Bayes (NB). Evaluation results show that SVM achieved the highest accuracy, precision, and recall, reaching 80%. On the other hand, [90] explored the detection of MiTM attacks by analyzing SDN traffic with a random forest classifier, achieving a notable F1-score of 98%.

The domain of anomaly detection has seen significant advancements with the introduction of novel methodologies. The ATLANTIC project [91] presents an approach utilizing a Support Vector Machine to detect and classify irregular traffic behaviors. This system functions in two stages: initially, traffic data is stored in structured tables, where entropy-based techniques identify deviations. Subsequently, SVM classifies the traffic as either normal or abnormal. Griffin [92] introduces an intrusion detection system that integrates AutoEncoders (AE) to detect both zero-day and known attacks in real time. The system employs a specialized feature extraction method to process sequential traffic data, achieving superior results in identifying Man-In-The-Middle attacks compared to conventional classifiers such as SVM and Random Forest. Additionally, [93] proposes an anomaly detection model designed for IoT networks within SDN. The framework consists of IoT devices at the base layer, followed by control and forwarding layers. Restricted Boltzmann Machines (RBM) facilitate network interactions, achieving an accuracy of 94%. An unsupervised anomaly detection technique is introduced in [94], where an Autoencoder minimizes reconstruction error before K-Means clustering categorizes traffic into normal and anomalous groups. In another approach, [95] applies the J48 algorithm within a structured input-processing-output system, efficiently handling packet security checks for the OpenFlow protocol. Evaluations using the KDD CUP 1999 dataset demonstrate its effectiveness when combined with Bagging techniques. Lastly, a hybrid IDS is presented in [96], combining flow-based and signature-based detection methods for malicious activity identification in SDN. The flow-based IDS operates in the control plane, utilizing an

ensemble feature selection strategy along with a multi-layer machine learning classifier (SVM + NB + C4.5). Meanwhile, the signature-based IDS in the data plane employs Snort's rule-based system. This hybrid model achieves 97.7% accuracy in flow-based detection and 95.26% in the rule-based approach.

Regarding application-layer flooding, VARMAN [97] is a framework that employs a machine learning pipeline combining Non-Symmetric Stacked Deep Autoencoder Learning with Random Forest classifiers to detect and mitigate application-layer flooding attacks. By analyzing request and response packets, the framework leverages protocol semantics to identify malicious activities. VARMAN enhances the system's ability to defend against various attack vectors, demonstrating the effectiveness of machine learning in strengthening application-layer security.

Finally, in the context of detecting DoS, Probe, U2R, and R2L attacks, [98] introduces a deep learning framework tailored for identifying flow-based anomalies in SDN environments. This framework utilizes multiple layers, including an input layer, hidden layers, and an output layer, achieving a detection accuracy of 75.75%. Expanding on this framework, [99] proposes a GRU-RNN-based Intrusion Detection System for SDN, achieving an accuracy of 89%, which represents a 13.25% improvement. Furthermore, [100] presents a technique that employs tree-based machine learning algorithms including RF, DT, and eXtreme Gradient Boosting (XGBoost) to detect DoS, Probe, U2R, and R2L attacks. Using only five features from the NSL-KDD dataset, this method reaches a high accuracy of 95.95%.

Table 1.12 provides an overview of existing research on network attack detection using machine learning methods. It compares various studies by examining the types of attacks identified, datasets employed, machine learning techniques applied, selected features, and the performance of the most effective ML approach.

Table 1.12: Summary of related work

Paper	**Target Attack**	**Datasets**	**ML Techniques**	**Selected Features**	**Best Performance**
[77]	DDoS (controller)	DARPA, InSDN, Simulated dataset	SVM, NB, FNN, DT, KNN, BT	Packet-In fluctuation	BT: 99.64% Acc.
[78]	DDoS	CICIDS2017, Simulated dataset	CNN, SVM, DNN, DT	Auto feature extraction	CNN: 98.98% Acc.
[79]	DDoS	Kaggle	CatBoost	Packet rate, Byte rate	Acc.: 98%, Recall: 82%, F1-score: 86%
[80]	DDoS	Simulated dataset (hping3)	SVM, KNN, NN, NB	12 features	KNN: 98.3% Acc.
[81]	LR-DDoS	Simulated dataset	CNN-LSTM, MLP, 1-Class SVM	12 features	CNN-LSTM: 99.998% Acc.
[82]	LR-DDoS	NSL-KDD, DARPA98, CAIDA, Simulated dataset	FM	4 features	Acc.: 95.80%

Continued on next page

Paper	Target Attack	Datasets	ML Techniques	Selected Features	Best Performance
[83]	LR-DDoS	CAIDA	WFL	78 features	Acc.: 98.85%, Sens.: 98.13%, F1-score: 94.21%
[84]	LR-DDoS	2017 CIC DoS dataset, Simulated dataset	J48, RF, SVM, MLP	40 features	MLP: 95.01% Acc.
[85]	Slow HTTP DDoS	Dataset generated through GNS3 SDN simulation	SVM	11 features	Acc.: 99.89%, AUC: 99.89%, TPR: 99.95%, FPR: 0.18%, FNR: 0.05%
[86]	DNS Flood	CICIDS	CNN-LSTM, SVM, ANN	78 features	CNN-LSTM: Acc.: 99.87%, Sens.: 99.85%, Prec.: 99.92%, Spec.: 99.90%, F1: 99.88%
[87]	DNS Flood	CICIDS2017 (normal), Scapy (malicious)	DT, RF, SVM, GBC, AdaBoost	12 features	SVM, GBC: $> 98\%$ Acc.
[88]	TCP-SYN, ICMP flood	CAIDA 2007, Simulated dataset	KNN, DT, NN	Ports/IP, entropy, ICMP count	KNN (K=9): $> 99\%$ Acc.
[89]	TCP DDoS	Real-world TCP traffic	NB, SVM, NN	Host count/sec, peak/off-peak	SVM: 80% Acc., 80% Prec., 80% Recall
[90]	MiTM	Collected SDN traffic data	RF	Not specified	F1-score: 98%
[91]	Anomalies	Collected SDN traffic data	SVM	IP, transport port	Acc.: 88.7%, Prec.: 82.3%
[92]	Anomalies	Open-source, Simulated dataset	Autoencoder	23 features	Acc.: 98%
[93]	Anomalies	KDD'99	RBM	41 features	Acc.: 94%
[94]	Anomalies	KDD'99	AE + K-Means	41 features	Acc.: 99%
[95]	Anomalies	KDD'99	J48 + Bagging	41 features	Acc.: 93.3%
[96]	Anomalies	NSL-KDD, Simulated dataset (Scapy)	SVM+NB+C4.5, Snort IDS	11 features (control plane)	Flow-IDS: 97.7%, Signature-IDS: 95.26% Acc.
[97]	App-layer flooding	CICIDS2017, Simulated dataset, Data from hackathon websites	Deep AE + RF	10 features	Acc. $> 98\%$
[98]	DoS, Probe, U2R, R2L	NSL-KDD	DNN	6 features	Acc.: 75.75%
[99]	DoS, Probe, U2R, R2L	NSL-KDD	GRU-RNN	6 features	Acc.: 89%

Continued on next page

Paper	Target Attack	Datasets	ML Techniques	Selected Features	Best Performance
[100]	DoS, Probe, U2R, R2L	NSL-KDD	DT, RF, XGBoost	5 features	XGBoost: 95.55% Acc.

1.7.4 Comparaison and Discussion

The results presented in Table 1.12 demonstrate the effectiveness of machine learning techniques in identifying different types of network intrusions. This analysis focuses on comparing models based on accuracy, as it is the most widely adopted metric for evaluating detection effectiveness in cybersecurity research [101, 102]

Several factors influence detection efficiency, including the type of attack, as different attacks exhibit varying patterns and complexities. The selected ML model and feature set also play a crucial role in determining how effectively threats are identified. For instance, in the case of DDoS attacks targeting SDN controllers, the Bagging Tree model achieved an accuracy of 99.64% by utilizing Packet-In message fluctuations as a key attribute [77]. Meanwhile, CNN achieved 98.98% accuracy in broader DDoS detection by leveraging automatic feature extraction, demonstrating its ability to process intricate data patterns [78]. These results indicate that Bagging Tree is more effective for detecting targeted attacks on SDN controllers, while CNN provides a robust solution for identifying a wider range of DDoS threats. The choice between these models depends on specific deployment requirements, balancing detection accuracy and computational efficiency based on the network's security needs.

For LR-DDoS attacks, hybrid approaches like CNN-LSTM have yielded near-perfect accuracy (99.998%) by capturing spatial and temporal dependencies within network traffic [81]. However, the superior accuracy of CNN-LSTM comes at the cost of increased computational complexity. In contrast, the Factorization Machine algorithm, which achieved 95.80% accuracy using just four features [82], provides a more resource-efficient alternative, making it suitable for environments with limited processing power.

Feature selection is essential for improving both classification accuracy and computational efficiency. In Slow HTTP DDoS detection, an SVM classifier achieved 99.89% accuracy using only 11 features [85]. Likewise, CNN-LSTM attained 99.87% accuracy in detecting DNS flood attacks but required a more extensive feature set of 78 attributes [86]. Although this larger feature space improves classification performance, it also increases computational overhead, reducing feasibility for real-time applications. Additionally, lightweight classifiers such as KNN and DT demonstrated strong detection capabilities, exceeding 98% accuracy for TCP-SYN and ICMP flood attacks while requiring only three features [88]. These findings highlight the necessity of tailoring feature selection to the characteristics of each attack type to maintain an optimal balance between detection effectiveness and computational efficiency.

Anomaly detection in SDN environments has been explored using various ML models with promising results. A hybrid technique integrating autoencoders with K-Means clustering achieved 99% accuracy [94], while another approach combining J48 with Bagging obtained 93.3% accuracy [95]. Although hybrid methods tend to yield higher accuracy, they also generally require greater computational

resources, which can affect their practicality in resource-limited SDN environments.

Choosing the right machine learning approach for network threat detection requires careful consideration of accuracy, computational efficiency, and feature selection. While CNN-LSTM models provide outstanding accuracy, their substantial computational demands make them more suitable for offline processing or high-resource settings. On the other hand, models such as SVM or KNN, when optimized with an appropriate feature selection strategy, can offer reliable performance with lower processing costs, making them ideal for real-time SDN scenarios. Efficient feature selection helps minimize computational burden without compromising detection reliability. Ultimately, validating models through real-world experimentation and optimization is essential to ensure their practical deployment in operational networks.

Choosing the right machine learning approach for network threat detection requires careful consideration of accuracy, computational efficiency, and feature selection. While CNN-LSTM models provide outstanding accuracy, their substantial computational demands make them more suitable for offline processing or high-resource environments. On the other hand, models such as SVM and KNN, when optimized with effective feature selection, can offer reliable performance with lower processing costs, making them ideal for real-time SDN scenarios. Efficient feature selection minimizes computational burden without compromising detection reliability. Ultimately, validating models through real-world experimentation and optimization is essential for ensuring their practical deployment in operational networks.

1.8 Future Research Directions

To fully realize the potential of ML solutions in SDN environments, several challenges need to be addressed:

- **Data Quality and Availability:** The performance of ML models heavily relies on the quality of the data used for training and evaluation. While some researchers create datasets in controlled environments, these often do not accurately represent real-world network traffic or attack patterns. A major challenge is the shortage of diverse, large-scale, labeled datasets from operational systems. This gap makes it difficult for ML models to effectively generalize across different network conditions and attack scenarios.

- **Detection of Low-Rate and Zero-Day Attacks:** Identifying low-rate attacks that mimic normal traffic patterns remains a challenge for many ML models. Zero-day attacks, which have no prior known patterns, present another significant obstacle for current models. Future research should explore more advanced ML techniques, including anomaly detection, to better address these complex threats.

- **Balancing Accuracy and Resource Use:** In SDN environments, a primary challenge is balancing the detection accuracy of ML models with their resource consumption. Developing efficient models that offer high detection accuracy while minimizing the use of essential network resources,

such as memory and processing power, is crucial.

- **Privacy-Preserving ML for Network Security:** With the increased use of ML, concerns about data privacy have emerged, especially regarding sensitive network traffic. Privacy-preserving ML techniques, such as federated learning, should be explored to enable secure collaboration between organizations while keeping sensitive network data protected.

1.9 Conclusion

The architecture of Software-Defined Networking offers a novel approach to network management but also introduces security challenges. The separation of the control and data planes enhances flexibility but also creates potential vulnerabilities, highlighting the need for specialized security measures. Intrusion Detection Systems powered by machine learning are essential for strengthening SDN security, as they can detect both known and emerging threats. Approaches like behavioral analysis and predictive modeling help create effective detection ML models. Selecting the right detection model requires balancing accuracy, resource consumption, and the choice of relevant features. While more complex models often deliver higher accuracy, they demand more computational resources, whereas simpler models with carefully selected features that are most relevant to detecting threats offer more efficient performance, making them better suited for real-time applications.

Achieving a balance between detection accuracy and computational efficiency is one of the major challenges in integrating machine learning into SDN. To address this, we propose three contributions that optimize this trade-off. The first approach combines K-Means and Naïve Bayes, providing an effective solution without significant computational overhead. The second builds on the first, replacing Naïve Bayes with Word2Vec and a neural network, which, with feature selection, maintains high accuracy while improving computational efficiency. The final combines ORB and a neural network, offering an alternative to the highly computationally demanding CNN.

Chapter 2

Flooding DDoS Detection in SDN Using K-Means and Naïve Bayes

Software-Defined Networking is a popular framework for network operators and service providers due to its programmability, high-level abstractions, and virtualization capabilities. However, it faces cybersecurity challenges, particularly in addressing flooding DDoS attacks, which exploit vulnerabilities and overwhelm systems with illegitimate traffic [103]. This type of attack is considered one of the most pressing and concerning threats in modern network security [104].

Within SDN architectures, OpenFlow switches play an integral role in effectively managing and forwarding traffic [105]. Malicious attackers often exploit the underlying design of these OpenFlow switches to launch flooding DDoS attacks by sending an overwhelming amount of packets that do not match the existing flow table entries. These unmatched packets are encapsulated into Packet-In messages and sent to the controller. This results in the overwhelming of the controller's processing resources, including critical components such as bandwidth, memory, and CPU. The excess amount of flooding traffic can lead to controller unresponsiveness, ultimately disrupting network functionality and severely impacting legitimate users trying to access essential services. Moreover, in this scenario, the switches themselves may become overwhelmed by the influx of malicious traffic, significantly reducing their ability to manage and handle legitimate network traffic effectively.

This chapter introduces an innovative approach that integrates K-Means clustering with Naïve Bayes classification for the efficient detection of DDoS attacks within SDN. By performing clustering at the feature level prior to classification, this technique enhances detection performance. Experimental assessments using the InSDN dataset demonstrate that this method outperforms not only Naïve Bayes but also several other recognized techniques. Additional testing with the CICIDS2017 dataset further confirms its effectiveness and suggests potential applications beyond SDN environments. Existing DDoS detection methods in SDN do not perform clustering at the feature level before classification. For instance, CNNs integrated with the Lion Optimization Algorithm (LOA) [106] and SVM with six-tuple features [107] do not incorporate this process. Similarly, methods such as P4-integrated classifiers [108], hybrid CNN-LSTM models for DNS flood detection [109], and dual-layer C4.5 decision tree frameworks [110] do not apply feature-level clustering. The proposed approach addresses this gap by integrating

clustering with classification, enabling a more detailed analysis of network traffic. This methodology enhances detection accuracy and provides a robust solution for mitigating flooding DDoS attacks in SDN environments, representing a significant step toward strengthening network security.

In the upcoming section, we discuss flooding DDoS attacks, their mechanisms, and traditional approaches for their detection.

2.1 Flooding DDoS Attacks in SDN

Flooding techniques in DDoS attacks are designed to overwhelm a network's bandwidth and exhaust its resources, causing it to become slow or unresponsive. These attacks take advantage of weaknesses in network infrastructure, such as limited link capacities, inefficient queuing systems, and insufficient resources at the end nodes (like servers or routers). Common attack methods include exploiting vulnerabilities in TCP and DNS amplification, which can amplify the volume of traffic sent to the target. Additionally, SYN flood attacks and other similar methods are used to flood the network with excessive traffic, leading to service interruptions and downtime.

2.1.1 Mechanisms of Flooding DDoS Attacks

Flooding DDoS attacks often involve the propagation of malware and the exploitation of vulnerable systems to create large botnets under the control of a central entity known as botmasters. These botnets send massive traffic to a target IP address, exhausting its resources and causing service disruption. Figure 2.1 illustrates a typical botnet DDoS attack.

Amplification techniques exacerbate the attack by increasing the size of the packet payload and sending spoofed traffic to widely deployed servers. This effectively turns legitimate servers into tools for launching attacks. Common amplification vectors, such as reflector-based UDP floods, use widely accessible servers to reflect and amplify traffic, generating traffic rates that can peak at hundreds of Gbps.

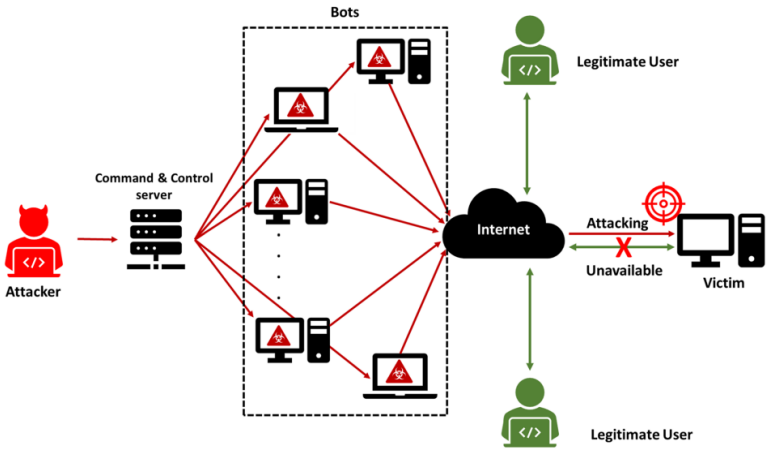


Figure 2.1: Botnet-based DDoS attack

2.2 Traditional Approaches for Flooding DDoS Detection

Several traditional methods have been employed to detect flooding DDoS attacks. Rule-based detection systems rely on predefined rules to identify attacks using specific Tactics, Techniques, and Procedures (TTPs) associated with known identifiers and attack signatures. Threshold-based rules trigger alerts when traffic loads exceed predefined limits, while time-based rules use temporal patterns and criteria to detect abnormal network behaviors. Although these strategies offer robust security, they are limited by their reliance on predefined rules, making it difficult to adapt to evolving attack tactics. Selecting appropriate thresholds is particularly challenging due to variations in attack intensity and legitimate traffic patterns. Other traditional approaches, such as inspecting TCP flags, are used to filter illegitimate traffic at the transport layer, helping to identify anomalous traffic patterns.

Despite their widespread use, these methods face significant challenges in achieving accurate and consistent detection performance under varying traffic conditions. One key limitation is the variability in network traffic, which makes it difficult to establish fixed detection rules. This variability arises from multiple factors, including fluctuations in attack intensity, the dynamic nature of legitimate user behavior, and differences in network configurations. As a result, detection systems face challenges in differentiating between normal and malicious traffic patterns, leading to classification errors. These errors manifest as false positives, where legitimate traffic is misidentified as an attack, or false negatives, where actual threats go undetected. This issue becomes even more critical in highly dynamic network environments like SDN, where centralized control and continuous traffic fluctuations further complicate real-time threat detection and adaptive response mechanisms.

Traditional detection techniques are also primarily designed to identify static attack patterns and known TTPs. As a result, they are less effective against modern flooding DDoS strategies, which exploit adaptive and evolving tactics to bypass detection mechanisms. These evolving strategies, coupled with the inherent limitations of traditional approaches, underscore the need for more flexible and dynamic detection frameworks.

Given the increasing sophistication and frequency of flooding DDoS attacks, enhancing detection frameworks is crucial. One promising direction involves adopting adaptive learning processes and machine learning algorithms to improve detection accuracy and response efficiency. Developing efficient learning-based frameworks could enable rapid adaptation to evolving attack tactics and dynamic network conditions. This approach holds the potential to provide a more robust and scalable solution to the challenges posed by modern flooding DDoS threats.

The next section provides background on K-Means and Naïve Bayes algorithms, highlighting their principles, applications, and limitations.

2.3 Background on K-Means and Naïve Bayes Algorithms

A. K-Means Clustering

K-Means clustering [111, 112] is a widely used method in data analysis, particularly valued for its application in unsupervised learning scenarios. The core idea is to group similar data points into distinct

clusters, with each cluster represented by its centroid. This algorithm aims to minimize variance within clusters while maximizing variance between different clusters, making it essential for identifying subgroups within complex datasets. This versatility makes K-Means clustering applicable across various domains, including marketing, finance, healthcare, and bioinformatics.

Figure 2.2 illustrates the process of the K-Means clustering algorithm. The algorithm begins by selecting a set of initial centroids, which significantly impact the clustering outcome. Centroids are then recalculated iteratively, and data points are assigned to clusters based on their proximity to these centroids, with Euclidean distance typically used as the metric for measurement. The iterative process continues until a convergence criterion is met, indicating that centroids have stabilized or that data points remain consistently assigned to the same clusters.

However, K-Means clustering has limitations. One significant drawback is that it does not guarantee a globally optimal solution; instead, it may converge to local optima depending on the initial selection of centroids. Additionally, the number of clusters (K) must be predefined, which can be challenging, particularly when the dataset's underlying structure is unclear. The algorithm's performance is also sensitive to outliers, which can distort clustering results and lead to less accurate outcomes.

Despite these limitations, K-Means clustering remains a fundamental tool in data science and machine learning due to its simplicity, computational efficiency, and effectiveness in a wide range of applications, from market segmentation to complex pattern recognition tasks in large datasets. Overall, K-means continues to be a crucial technique relied upon by analysts and researchers for extracting valuable insights from data.

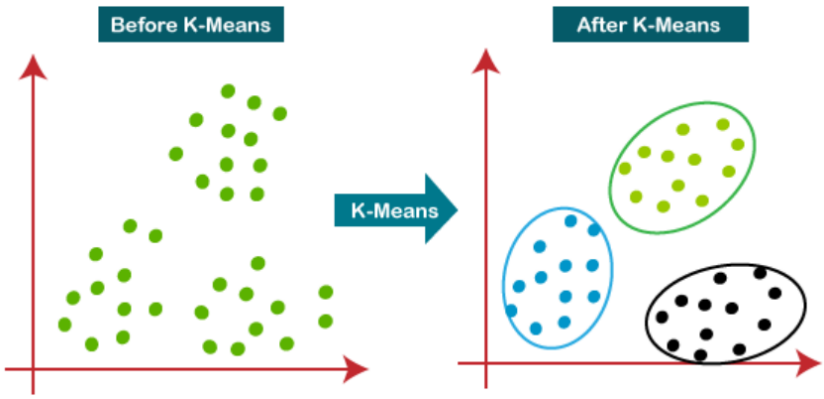


Figure 2.2: K-Means clustering process overview

B. Naïve Bayes

Naïve Bayes classifiers are a group of probabilistic models based on Bayes' theorem, widely used in machine learning for classification tasks. These models simplify computations by assuming that all features are conditionally independent given the class label, which is the "Naïve" aspect of their design. This assumption significantly reduces computational complexity, making them particularly well-suited for large datasets and scenarios where feature interactions are minimal. The simplicity and efficiency of Naïve Bayes enable it to handle tasks requiring rapid model training and deployment.

The core principle of Naïve Bayes lies in Bayes' theorem, which calculates the probability of a class given the observed features. The formula:

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} \tag{2.1}$$

represents the posterior probability $P(c|x)$, where $P(x|c)$ is the likelihood of the features given the class, $P(c)$ is the prior probability of the class, and $P(x)$ is the evidence probability. By assuming conditional independence, Naïve Bayes simplifies the likelihood $P(x|c)$ into a product of individual feature probabilities.

To classify an instance, Naïve Bayes selects the class c^* that maximizes the posterior probability:

$$c^* = \arg\max_c P(c|x) = \arg\max_c P(x|c)P(c) \tag{2.2}$$

This reduction in complexity allows the model to compute probabilities efficiently, directly contributing to its speed in training and deployment, particularly with large-scale data. In this way, Naïve Bayes can quickly classify instances by selecting the most probable class, making it an effective tool for time-sensitive tasks.

Naïve Bayes classifiers come in various forms to accommodate different types of data. *Gaussian Naïve Bayes* assumes that features follow a Gaussian distribution, making it suitable for continuous data. *Multinomial Naïve Bayes* is tailored for discrete data and is commonly used in text classification tasks, such as spam detection and sentiment analysis. *Bernoulli Naïve Bayes*, on the other hand, works with binary features and is effective in determining the presence or absence of specific attributes, such as keywords in a document.

The applications of Naïve Bayes are diverse, spanning fields like spam detection, sentiment analysis, document categorization, medical diagnosis, and sequence recognition. For instance, in spam email detection, the classifier uses feature vectors derived from email content to identify unsolicited messages. In sentiment analysis, it classifies text data to determine whether the sentiment is positive, negative, or neutral, making it a popular tool in analyzing customer feedback and social media content. Additionally, Naïve Bayes is instrumental in document categorization, helping organize and retrieve information efficiently, and in medical diagnosis, where it assists in predicting diseases based on patient symptoms.

While Naïve Bayes excels in simplicity and speed, its reliance on the independence assumption can be a limitation. This assumption may not hold in datasets with correlated features, potentially affecting the model's accuracy. However, its ability to perform well in many real-world scenarios, even with this limitation, underscores its robustness. Naïve Bayes remains a foundational tool in machine learning, valued for its straightforward implementation and adaptability. Ongoing research continues to explore hybrid models that extend its capabilities, making it suitable for increasingly complex applications.

2.4 Experimental Setup

This section presents a comprehensive overview of the experimental setup, including the datasets used, the selected features, and the proposed methodology. It also outlines the evaluation procedure and the

metrics used to assess the model's performance.

2.4.1 Experiment Datasets

We evaluated our model using the InSDN dataset, specifically designed for analyzing SDN-related attacks. It provides diverse attack scenarios tailored to SDN environments, making it suitable for evaluating SDN attack detection techniques. The dataset includes various flooding DDoS attacks, such as TCP-SYN Flood, UDP Flood, and ICMP Flood, executed using the Hping3 tool. To construct our experimental dataset, we combined the Normal and OVS datasets in CSV format. As a result, the final dataset consists of seven categories: Normal, BFAs, BotNet Attacks, DoS, DDoS, Probes, and Web Attacks. For classification, we merged all non-DDoS classes into a single category labeled "Others," resulting in a binary classification dataset with two classes: DDoS and Others. The instance distribution is illustrated in Figure 2.3. Additionally, we excluded the features 'Src IP', 'Src Port', 'Dst IP', and 'Dst Port' to reduce the risk of overfitting. The experimental dataset is named InSDN_DDoS_Exp [113].

We further tested our model with the CIC-DDoS2017 dataset, specifically utilizing the CSV file "Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv." This dataset includes DDoS attacks generated with the LOIC (Low Orbit Ion Cannon) tool, which floods targets with excessive traffic to disrupt accessibility. It contains a disproportionately high number of DDoS attack instances compared to benign traffic, which may not fully reflect real-world conditions. To refine the dataset, we removed infinite values and created an experimental version with a more balanced distribution of attack and benign instances [114]. The instance distribution for this dataset is also shown in Figure 2.3.

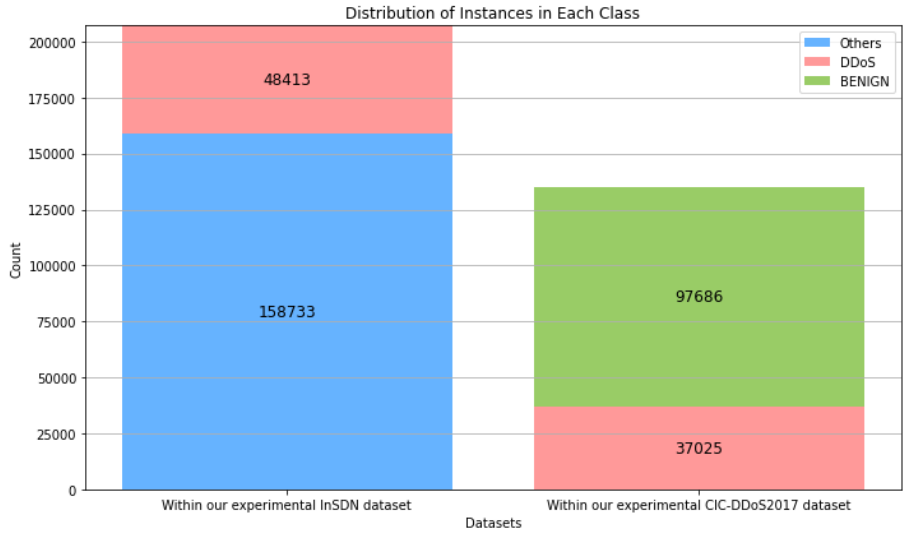


Figure 2.3: Class instance distribution of 'DDoS' and 'Others'

2.4.2 Features Employed in Our Experiments

This section outlines the features used in our study and describes the procedure for feature selection. We applied the SelectKBest technique from the scikit-learn library to identify the most critical features in the dataset. This method evaluates multiple values of 'k,' which specifies the number of features to be

selected. For each 'k,' it calculates the chi-squared (chi2) statistic between individual features and the target variable, selecting the 'k' features with the highest chi2 scores. A cross-validation approach with a Random Forest classifier was then used to determine the optimal 'k' value. The feature subset with the highest average cross-validation score across different 'k' values was chosen as the final set. This approach ensures that the selected features enhance the model's predictive performance and accuracy.

The final subset of features, derived from our experiments on the InSDN and CIC-DDoS2017 datasets, is detailed in Table 2.1. The table ranks features based on their relevance. This approach ensures a targeted focus on the most impactful features. During the analysis of the CIC-DDoS2017 dataset, the 'Destination Port' feature was omitted to minimize the risk of overfitting and emphasize features more directly related to DDoS attack detection.

Table 2.1: Extracted a subset of features from our experimental datasets, InSDN and CIC-DDoS2017

No.	Feature	No.	Feature
InSDN Dataset			
1	Protocol	7	SYN Flag Cnt
2	Flow Duration	8	PSH Flag Cnt
3	Flow Pkts/s	9	ACK Flag Cnt
4	Bwd PSH Flags	10	Down/Up Ratio
5	Bwd Pkts/s	11	Init Bwd Win Byts
6	FIN Flag Cnt		
CIC-DDoS2017 Dataset			
1	Fwd Packet Length Max	15	Max Packet Length
2	Fwd Packet Length Min	16	Packet Length Mean
3	Fwd Packet Length Mean	17	Packet Length Std
4	Fwd Packet Length Std	18	Packet Length Variance
5	Bwd Packet Length Max	19	SYN Flag Count
6	Bwd Packet Length Min	20	PSH Flag Count
7	Bwd Packet Length Mean	21	URG Flag Count
8	Bwd Packet Length Std	22	Down/Up Ratio
9	Bwd IAT Total	23	Average Packet Size
10	Bwd IAT Mean	24	Avg Fwd Segment Size
11	Bwd IAT Std	25	Avg Bwd Segment Size
12	Bwd IAT Max	26	Subflow Fwd Bytes
13	Fwd PSH Flags	27	Init_Win_bytes_backward
14	Min Packet Length		

2.4.3 Proposed Approach

Figure 2.4 illustrates the DDoS attack detection methodology, integrating multiple K-Means clustering models with a Gaussian Naïve Bayes classifier. The approach is structured into the following phases:

- **Training multiple K-Means and Naïve Bayes models**: A set of K-Means models is trained iteratively, with each set corresponding to a specific number of clusters (k) within a predefined

range. For each k value, separate K-Means models are trained in parallel for individual features in the dataset, ensuring that clustering is performed independently for each column rather than on the entire dataset. Then, each feature in the training set is grouped into up to k distinct clusters using its corresponding trained K-Means model, identifying patterns unique to that column. The resulting clusters are then used as input to train a Gaussian Naïve Bayes classifier to distinguish between two classes in a binary classification task.

- **Choosing the optimal model**: Each iteration, corresponding to a specific k value, produces a combination of multiple trained K-Means models and a Gaussian Naïve Bayes classifier. These combinations are assessed using the validation dataset to measure their accuracy. The configuration that achieves the highest accuracy is considered the optimal model.

- **Evaluating the optimal model**: The optimal model is assessed using the testing dataset, which is first clustered using the trained K-Means models. These clusters are then passed as inputs to the trained Gaussian Naïve Bayes classifier to perform binary classification.

Algorithm 1 outlines the construction process of the optimal model, demonstrating how K-Means clustering and the Naïve Bayes classifier collaborate to improve classification accuracy.

Algorithm 1: Proposed DDoS detection approach

Input: Training dataset, validation dataset, testing dataset
Output: Best-performing DDoS detection model

Initialize best_KMeans_models;
Initialize best_NB_model;
Initialize best_accuracy $= 0$;
Define X as the matrix of feature values and y as the vector of target values;
Step 1: Train K-Means models KMeans_models$_k$ and Naïve Bayes model NB_model$_k$;
for *each k in a predefined range* **do**
    Train K-Means models KMeans_models$_k$ using Parallel Processing and assign discrete categories;
    for *each $column$ in X_{train}* **do**
        Train K-Means model KMeans_model$_{k,column}$ on $X_{\text{train}}[column]$;
        Append KMeans_model$_{k,column}$ to KMeans_models$_k$;
    end for
    Assign clusters to X_{train} using KMeans_models$_k$;
    Train Naïve Bayes model NB_model$_k$ using cluster assignments;
    Step 2: Evaluate on validation set;
    Evaluate NB_model$_k$ on X_{val};
    Compute validation accuracy accuracy$_k$;
    if *accuracy$_k$ $>$ best_accuracy* **then**
        Update best_accuracy to accuracy$_k$;
        Update best_KMeans_models to KMeans_models$_k$;
        Update best_NB_model to NB_model$_k$;
    end if
end for
Step 3: Evaluate the best model on testing set;
Apply best_KMeans_models to X_{test} for clustering;
Apply best_NB_model for classification on clustered data;

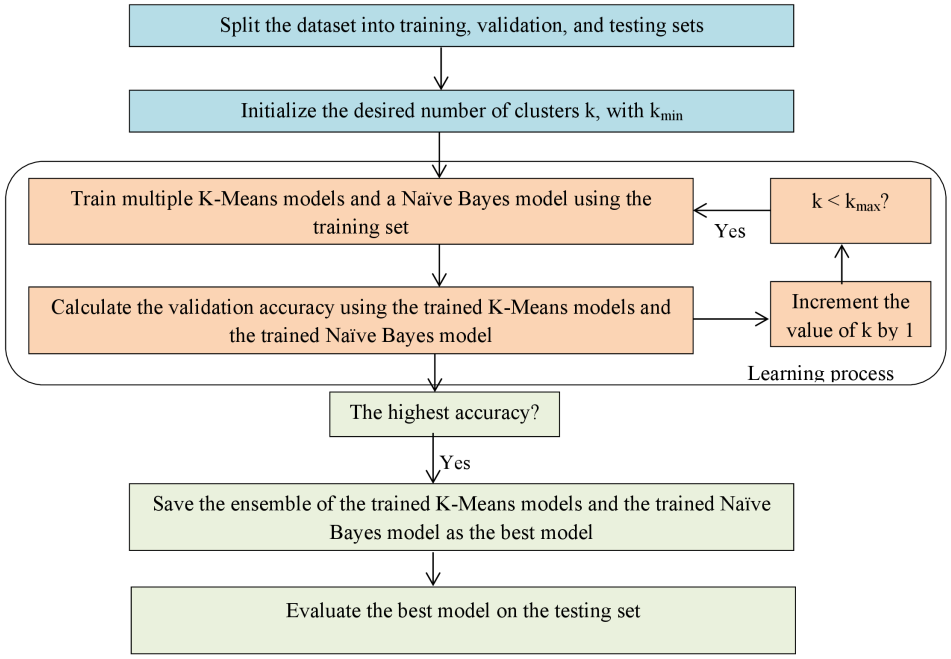


Figure 2.4: Flowchart of the proposed DDoS detection approach

2.4.4 Evaluation Process and Metrics

We tested our approach by varying the desired number of clusters (k), where each model consists of multiple K-Means models and a Naïve Bayes classifier. The model with the highest accuracy is considered optimal. Initially, we compared its performance to that of a Naïve Bayes classifier using the InSDN dataset. Afterwards, we evaluated its performance on the CIC-DDoS2017 dataset. The evaluation employed standard metrics, including accuracy, precision, recall, and F1-score, to measure the model's effectiveness.

2.5 Results and Discussion

In this section, we present the results of our experimental evaluation. We first analyze the accuracy of both the Naïve Bayes model and our proposed model across different k values. Next, we conduct a detailed analysis of the classification report for the model that achieved the highest accuracy. The results are presented in Figures 2.5 to 2.8, highlighting key findings and their interpretations.

2.5.1 Result on InSDN Dataset

As shown in Figure 2.5, both models demonstrate strong accuracy. Our approach closely mirrors Naïve Bayes, achieving an impressive 99.9742% accuracy across various k values. With k=32, our model surpasses Naïve Bayes slightly, reaching 99.9839% accuracy, reflecting a modest improvement of 0.0097%. Surpassing the accuracy of the NB model, which already achieved high accuracy, even slightly, demonstrates the effectiveness of our model and suggests that it can yield promising results. Furthermore, Figure 2.6 reveals that with k = 32, our model delivers solid precision, recall, and F1-score values of

approximately 99.9455%, 99.9863%, and 99.9659%, respectively, demonstrating its effectiveness in accurately identifying positive instances and reducing misclassifications.

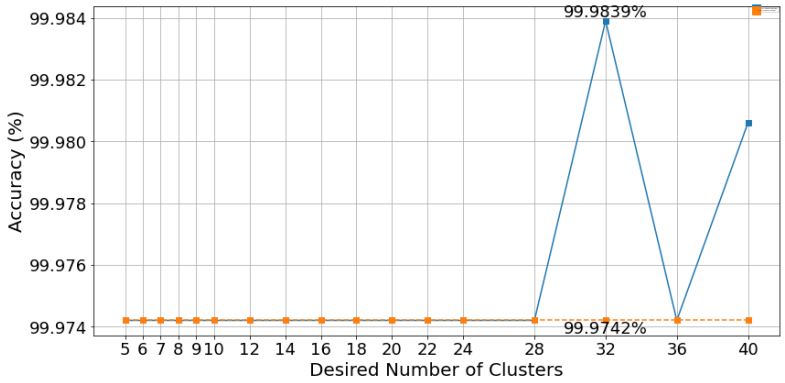


Figure 2.5: Accuracy across desired number of clusters within our experimental InSDN dataset

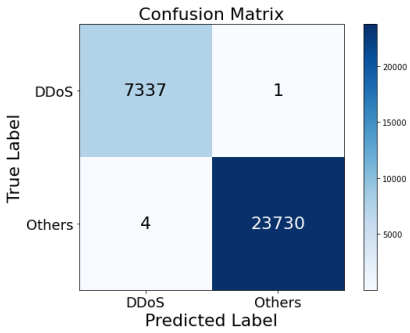


Figure 2.6: Confusion matrix within our experimental InSDN dataset

2.5.2 Result on CIC-DDoS2017 Dataset

As illustrated in Figure 2.7, the proposed model starts with lower accuracy compared to the Naïve Bayes model. However, as the desired number of clusters increases, our model gradually improves. Around k = 24, our model surpasses the Naïve Bayes model and continues to outperform it. At k = 86, our model achieves an accuracy of 99.7030%, significantly exceeding Naïve Bayes' accuracy of 96.3774%, resulting in an increase of 3.3256%. This demonstrates the enhanced predictive performance of our model. Additionally, Figure 2.8 shows that with k = 86, our model reaches a precision of 99.2487%, meaning 99.2487% of predicted positives are correctly classified, and a recall of 99.5693%, indicating its ability to identify 99.5693% of actual positives. Therefore, the F1-score, balancing precision and recall, reaches 99.4088%.

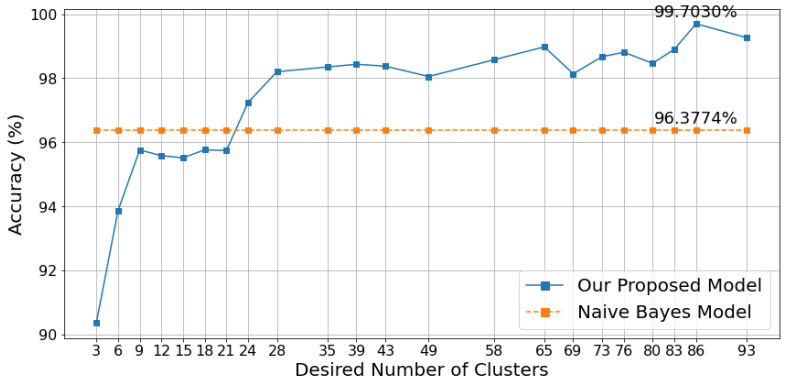


Figure 2.7: Accuracy across desired number of clusters within our experimental CIC-DDoS2017 dataset

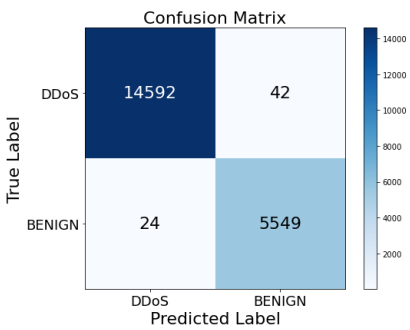


Figure 2.8: Confusion matrix within our experimental CIC-DDoS2017 dataset

2.5.3 Computational Overhead

The computational overhead of this approach remains low, as clustering is applied at the feature level, and Naïve Bayes requires minimal computational resources. K-Means clustering is performed independently in a one-dimensional space, with all K-Means models trained in parallel, further reducing clustering complexity and training time. Additionally, Naïve Bayes enables rapid model training and deployment, making it well-suited for real-time applications.

As shown in Figures 2.9 and 2.10, training time scales with the number of clusters, with processing durations ranging from 3.28 seconds (5 clusters) to 29.59 seconds (40 clusters) in the InSDN dataset and from 5.20 seconds (3 clusters) to 124.06 seconds (93 clusters) in the CIC-DDoS2017 dataset. Computational efficiency and training time can be improved through optimizations such as Mini-Batch K-Means, which reduces per-iteration updates, and parallel processing with Graphics Processing Units (GPUs), which speeds up execution on large-scale datasets.

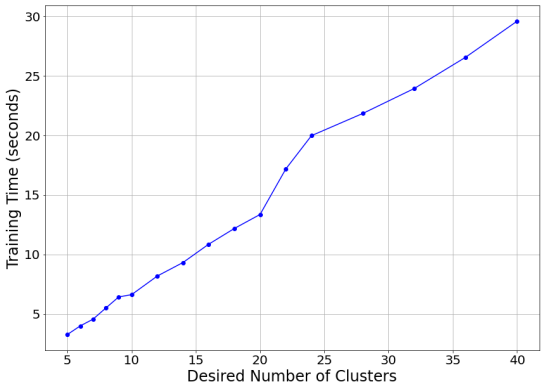


Figure 2.9: Training time across desired number of clusters within our experimental InSDN dataset

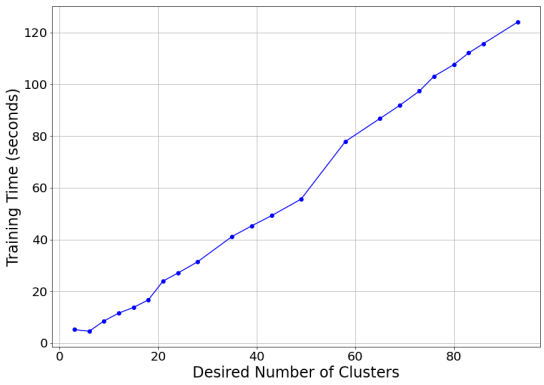


Figure 2.10: Training time across desired number of clusters within our experimental CIC-DDoS2017 dataset

2.5.4 Comparison

Table 2.2 compares our proposed approach with existing research, showcasing the superior performance of our hybrid model. For instance, [115] achieves 99.64% accuracy with BT, while our model reaches 99.98% on the InSDN dataset. Similarly, [78] reports 98.98% accuracy with CNN, whereas our method attains 99.70% on the CIC-DDoS2017 dataset. Other studies such as [116] and [80] report maximum accuracies of 98.7% and 98.3% with CART and KNN, respectively, which our model surpasses. [117] reports 96% precision and 98% recall using KNN, whereas our method achieves higher precision and recall of 99.95% and 99.99%, respectively, on the InSDN dataset. Furthermore, [118] achieves 98.907% accuracy using a combination of KPCA, SVM, and GA on the NSL-KDD and self-generated datasets. In contrast, our model significantly exceeds this with an accuracy of 99.98% on the InSDN dataset and 99.70% on the CIC-DDoS2017 dataset. Moreover, [89] reports 80% accuracy, precision, and recall using SVM on real-time TCP traffic data. Our model demonstrates superior performance with 99.98% accuracy, 99.95% precision, and 99.99% recall on the InSDN dataset. Lastly, [88] shows that KNN achieves an accuracy rate exceeding 99%, while our model achieves even higher accuracy. Despite the

promising results of our approach, reducing the computational overhead introduced by training multiple K-Means models is crucial, as it can pose challenges in large-scale scenarios. However, implementing efficient parallelization strategies and leveraging distributed computing frameworks can help address these challenges, making our model more suitable for large-scale deployment.

Table 2.2: Comparison of various DDoS detection models

Reference	Dataset	ML Techniques	Performance
[115]	DARPA, InSDN, and self-generated dataset using simulation	SVM, GLM, NB, FNN, DT, KNN, BT	BT achieves the highest accuracy of 99.64%
[78]	CICIDS2017 and self-generated dataset	CNN, SVM, DNN, DT	CNN achieves 98.98% accuracy (4.25% to 8.20% better than others)
[116]	Self-generated dataset	QDA, GNB, KNN, CART	CART achieved the highest accuracy of 98.7%, with all methods exceeding 95%
[80]	Self-generated dataset using hping3	SVM, KNN, NN, NB	KNN achieved the highest accuracy rate of 98.3%
[117]	Self-generated dataset by simulating different DDoS attacks in SDN testbed	KNN, ANN	KNN: 96% precision, 98% recall; ANN: 90% precision, 86% recall
[118]	NSL-KDD and self-generated dataset using NS2	KPCA, SVM, GA	KPCA+SVM+GA model achieved 98.907% accuracy
[89]	Dataset collected in real-time from TCP traffic	NB, SVM, NN	SVM shows 80% accuracy, 80% precision, and 80% recall. SVM is considered the better choice
[88]	CAIDA 2007 and self-generated dataset	KNN, DT, NN	Accuracy is above 98% for all three ML techniques; KNN achieved over 99% accuracy when K=9
Proposed model	InSDN and CIC-DDoS2017	Hybrid K-Means and Naïve Bayes Model	In InSDN dataset: 99.98% accuracy, 99.95% precision, 99.99% recall, 99.97% F1 score. In CIC-DDoS2017 dataset: 99.70% accuracy, 99.25% precision, 99.57% recall, and 99.41% F1 score

2.6 Conclusion

This chapter explores the detection of flooding DDoS attacks in SDN environments, emphasizing the challenges posed by excessive traffic that aims to overwhelm network resources. These attacks necessitate comprehensive security strategies for effective detection and mitigation. To address these challenges, we propose a model that combines K-Means clustering with Gaussian Naïve Bayes classification. This combination seeks to enhance the detection accuracy of flooding DDoS attacks by identifying subtle patterns in network traffic. We assess the effectiveness of our approach using real-world datasets, including InSDN and CIC-DDoS2017. Our experiments demonstrate that by adjusting the desired number of clusters in the K-Means algorithm, the model achieves impressive detection accuracy, effectively identifying flooding DDoS attacks.

This approach demonstrates exceptional accuracy in detecting flooding DDoS attacks; however, it primarily focuses on a specific attack type. The next contribution builds on this approach by further exploring the effectiveness of one-dimensional K-Means clustering, proposing a more advanced and comprehensive traffic classification method. This method integrates multiple K-Means clustering with Word2Vec for feature extraction, followed by neural network classification.

Chapter 3

Enhancing SDN security with a feature-based approach using multiple k-means, Word2Vec, and neural network

This chapter introduces a novel approach to traffic classification by leveraging clustering and feature extraction techniques to enhance detection accuracy. Our methodology combines multiple K-Means models with Word2Vec for feature extraction, followed by neural network classification. This structured approach improves the analysis of traffic patterns, enabling more precise anomaly detection. We evaluate our method using the InSDN dataset, demonstrating its ability to achieve higher accuracy than a baseline neural network model. Further testing on the CIC-DDoS2019 dataset confirms its effectiveness in distinguishing normal traffic from various types of DDoS attacks, highlighting its adaptability across different scenarios.

Several studies have explored various techniques to improve detection capabilities. For example, the Range Supported Bit Vector (RSBV) has been used for packet classification [119], while SADM-SDNC utilizes the NetFlow protocol for anomaly detection [120]. Approaches like DeepIDS and Deep-Discovery IDS employ DNNs and ANNs, respectively, to identify abnormal activities [121], [122]. Furthermore, machine learning methods such as Random Forest, KNN, and Decision Trees have been applied to analyze network traffic [123]. While these methods contribute to more efficient detection mechanisms, many rely solely on clustering or classification without integrating feature extraction techniques to enhance traffic analysis. Despite these advancements, a gap remains in effectively combining clustering with feature extraction to improve classification accuracy.

Our approach addresses this gap by integrating K-Means clustering with Word2Vec-based natural language processing for feature extraction, enabling a more comprehensive traffic analysis. While existing methods often apply clustering to entire datasets [124, 125] without leveraging feature extraction, our method enhances classification by capturing distinctive traffic characteristics. This refined analysis improves traffic classification, leading to higher accuracy.

The next section discusses the limitations of traditional traffic classification methods and introduces machine learning techniques for improved traffic analysis.

3.1 Machine-Learning-Based Traffic Classification

Traditional methods of traffic classification in computer networks, such as port-based classification, deep packet inspection (DPI), and statistical-based techniques, have long been used to differentiate traffic types [126]. However, these rule-based approaches face substantial challenges in SDN environments. Port-based classification, which relies on fixed port numbers to identify applications and services, becomes unreliable as modern applications increasingly use dynamic or non-standard ports. DPI provides a detailed examination of packet contents but is computationally intensive, may introduce significant latency, and faces scalability issues as network traffic increases. Maintaining up-to-date signatures for application identification is particularly challenging with encrypted traffic [127]. Similarly, statistical-based classification, which analyzes the statistical properties of traffic streams, is less resource-intensive but requires mapping clusters to applications, creating challenges for accurate traffic classification [128, 129]. These limitations underscore the need for more advanced and adaptable traffic classification techniques in SDN environments to effectively manage the dynamic and complex nature of modern network traffic.

Machine learning algorithms have become powerful tools for improving network traffic classification. Unlike traditional methods, machine learning-based approaches adapt to the dynamic and evolving nature of network traffic. By leveraging techniques such as supervised learning, unsupervised learning, and deep learning, these algorithms can analyze complex traffic patterns, detect anomalies, and classify traffic types with high accuracy. In SDN environments, where network configurations and traffic flows change rapidly, machine learning provides the flexibility to handle diverse traffic patterns, including encrypted and non-standard traffic. Additionally, ML-based methods overcome the limitations of port-based classification and the need for manual signature updates in DPI by automatically learning from data. As a result, machine learning provides a scalable and efficient solution for traffic classification, offering enhanced performance in SDN-based networks.

The next section explores neural networks and Word2Vec, covering the structure, training process, and limitations of neural networks, as well as Word2Vec's CBOW and Skip-gram models for word representation.

3.2 Overview on Word2Vec and Neural Networks

3.2.1 Neural Networks Overview: Functionality and Limitations

Artificial Neural Networks (ANNs) represent sophisticated computational models created to emulate the intricate patterns of neuronal interconnections existing within the human brain. These networks excel in their ability to capture and analyze complex, nonlinear patterns that exist within various types of data. Throughout the years, ANNs have been effectively applied across a wide range of diverse domains, including but not limited to image processing, speech recognition, recommendation systems, advanced robotics, and autonomous driving technologies, consistently achieving remarkable state-of-the-art results in numerous applications.

A. Basic Structure of a Neural Network

A typical neural network structure consists of three distinct types of layers, each serving a specific role in processing information:

- **Input Layer**: This layer accepts and processes the raw data that enters the network. It serves as the gateway for data to flow into the network for further processing.
- **Hidden Layers**: These layers perform essential computations and feature extraction. They enable the network to learn complex patterns and relationships within the data by processing and transforming information through multiple layers of neurons.
- **Output Layer**: The main purpose of the output layer is to generate the final result, which could be a prediction or classification based on the processed data.

Each layer comprises neurons connected by weighted links. The flow of information through these layers enables the network to learn and represent intricate data patterns. Figure 3.1 illustrates a neural network architecture with an input layer consisting of two nodes, a hidden layer with two neurons, and an output layer with a single neuron.

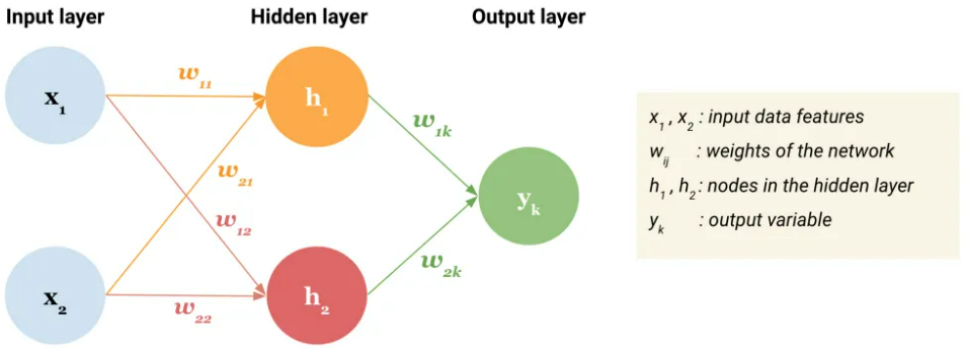


Figure 3.1: Sample neural network structure

Activation functions are applied to the outputs of neurons within the hidden layers and, in some cases, the output layer. Common activation functions include:

- **Sigmoid**: Maps input to a range between 0 and 1.
- **Tanh**: Maps input to a range between -1 and 1.
- **ReLU (Rectified Linear Unit)**: Outputs the input directly if positive; otherwise, it outputs zero.

The choice of activation function in the output layer depends on the specific task and desired output range. For example, in binary classification tasks, the sigmoid activation function is commonly used in the output layer to produce a probability value between 0 and 1. The application of activation functions in the hidden layers allows the network to learn and represent intricate data patterns, enhancing its ability to model complex relationships within the data.

Weight adjustments are made carefully during the training phase to minimize the error between the network's prediction and the actual desired output. This process involves calculating gradients and updating the network's weights to improve its accuracy. Training consists of two essential steps:

- **Forward Propagation**: In this step, input data is passed systematically through the network, allowing it to generate an initial output based on its current weights and biases.
- **Backward Propagation**: Here, the error is calculated by comparing the predicted output with the actual target output. Gradients are then computed, enabling the network to update its weights effectively. Typically, optimization algorithms such as gradient descent are employed for this purpose.

This iterative process is fundamental to the learning process, as it helps the network refine its performance over time, leading to increasingly accurate predictions for future tasks.

B. Neural Networks Limitations

Despite their impressive capabilities and widespread applications, neural networks come with several notable limitations:

- **Data Requirements**: Neural networks often require large amounts of labeled data for effective and efficient training. This can be a significant challenge in practical scenarios where obtaining sufficient labeled data is difficult.
- **Computational Resources**: Training deep neural networks is computationally intensive and demands substantial resources, including significant processing power and time. This may not always be readily available, particularly for resource-constrained environments.
- **Interpretability**: Neural networks are often considered "black boxes," making it challenging for practitioners and researchers to understand how decisions are made and how specific conclusions are reached. This lack of transparency can hinder the application in certain domains where interpretability is crucial.
- **Overfitting**: Without proper regularization or management during the training process, neural networks are prone to overfitting. This leads to models that perform well on the training data but fail to generalize to new, unseen data.

3.2.2 Word2Vec: Architecture, Applications, and Challenges

Word2Vec [130, 131] is a widely recognized and influential neural network model extensively used in the field of natural language processing. This innovative model transforms textual information into numerical representations, enabling efficient processing by various machine learning algorithms. Unlike traditional methods, such as the "bag of words," which fail to capture the intricate semantic relationships between words, Word2Vec generates "word embeddings." These embeddings are vector-based representations that provide deeper insights and connections among a diverse set of words. Word2Vec's ability

to numerically represent contextual information allows for a broad range of computations, predictions, and classifications applicable to various NLP tasks.

Word embeddings are continuous vectors situated in a high-dimensional space. These embeddings can be derived from statistical co-occurrence matrices or through advanced neural network techniques, effectively capturing syntactic, semantic, and even emotional nuances conveyed by words based on their proximity within the expansive vector space. For instance, semantically similar words cluster closely together, while those with contrasting meanings are spaced further apart.

Word2Vec operates using two primary architectures: Continuous Bag of Words (CBOW), illustrated in Figure 3.2, and Skip-gram, depicted in Figure 3.3, each serving a distinct purpose in capturing word relationships.

A. Continuous Bag of Words Architecture

In the CBOW architecture, the model predicts a target word based on the surrounding context words. For example, in the sentence, ”The cat sits on the mat,” if the context is {”The,” ”cat,” ”on,” ”the”}, CBOW predicts the missing word, ”sits.” This architecture is efficient for datasets with frequent word occurrences and is particularly useful for generating embeddings that prioritize common words over rarer terms.

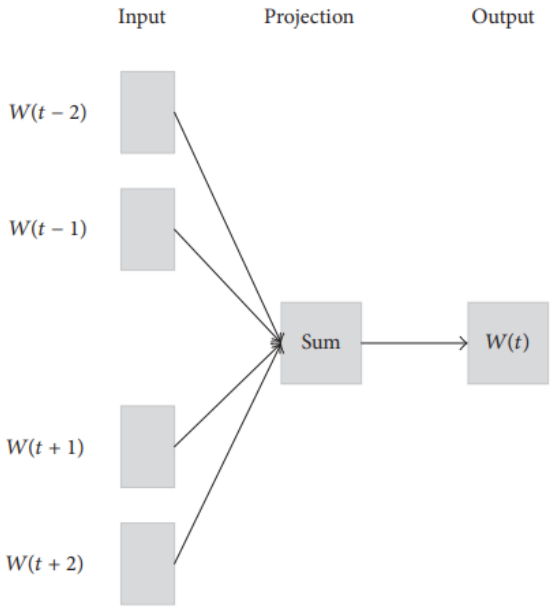


Figure 3.2: CBOW model architecture

B. Skip-gram Architecture

In contrast, the Skip-gram architecture predicts context words based on a specific target word. For instance, given the target word ’sits’ in the sentence, ’The cat sits on the mat,’ and using a window size of 2, the model predicts context words like ’The,’ ’cat,’ ’on,’ and ’the.’ Skip-gram is particularly effective at capturing rare word relationships, as it prioritizes learning from less frequent co-occurrences. This

approach ensures the creation of robust word embeddings that represent both common and rare words. By adjusting the context window size (e.g., including two or more words before and after a target word), Word2Vec can fine-tune the embeddings to better capture semantic and syntactic relationships.

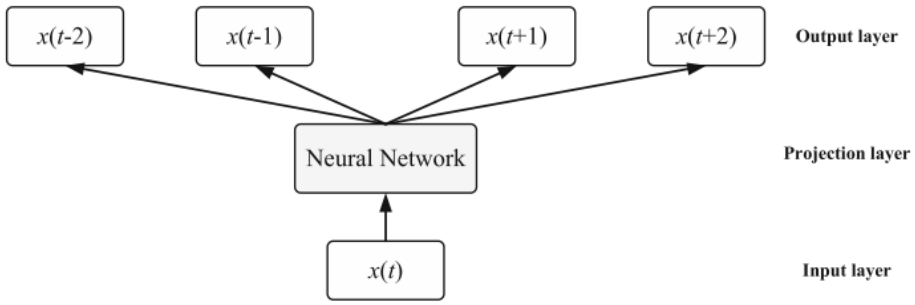


Figure 3.3: Skip-gram model architecture

Word2Vec has broad applications across various fields. In text classification, it improves feature extraction by converting words into embeddings. In translation, it enhances language models and dictionary generation through collaborative data collection. For sentiment analysis, these embeddings help assess consumer opinions from social media and online reviews. Additionally, Word2Vec strengthens recurrent neural networks and other deep learning architectures used in machine learning.

Industries such as healthcare and finance utilize Word2Vec for voice digitization, improving decision-making by analyzing spoken transcripts. It also enhances feature engineering by efficiently handling large text datasets, outperforming traditional one-hot encoding. In text summarization, Word2Vec extracts key insights, improving information retention.

In the domain of information retrieval, Word2Vec significantly transitions from conventional keyword searches to a deeper understanding of word relationships, thereby improving search query accuracy and relevance. This evolution leads to more effective search engine functionality.

Despite these strengths, Word2Vec faces several challenges. Data sparsity can result in suboptimal embeddings, as limited training data may fail to capture nuanced details present in smaller datasets. Additionally, its dependence on previously encountered words during training creates difficulties in integrating out-of-vocabulary words, limiting its effectiveness with new or rare terms. Domain adaptation remains a significant challenge, as word meanings may vary across different contexts, making it difficult to maintain meaningful and relevant embeddings in specific fields. Other challenges include interpreting subtle sentiments in text and sustaining robust representations amidst diverse linguistic patterns or across various domains.

While these challenges persist, ongoing research into Word2Vec and word embeddings highlights their critical role in the broader landscape of NLP. Future studies should focus on improving the interpretability and robustness of these models, particularly in domain-specific contexts, while enhancing their adaptability to accommodate new linguistic forms through interdisciplinary collaboration. The commercial applications of Word2Vec, especially in advertising, healthcare, and environmental research, underscore its continued relevance and significant potential for industry growth. As research advances, Word2Vec's impact will continue to evolve, deepening our understanding and utilization of language

data in increasingly sophisticated ways.

3.3 The Datasets Employed and the Proposed Approach

In this section, we introduce an innovative methodology for traffic analysis that employs K-Means clustering, Word2Vec, and neural network models. The objective of this approach is to improve the precision of anomaly detection and traffic classification in Software-Defined Networking environments. Furthermore, we will elaborate on the datasets utilized in our experiments, including the InSDN dataset, which is dedicated to SDN-related attacks, as well as the CIC-DDoS2019 dataset.

3.3.1 Datasets

In this study, we use InSDN_DDoS_Exp.rar [132], which is derived from the InSDN dataset, specifically designed for SDN attack detection. The distribution of instances in our experimental InSDN dataset is shown in Figure 3.4. Our evaluations also extend to the CIC-DDoS2019 dataset, which classifies traffic into five DDoS attack types alongside benign traffic. To prevent overfitting, we removed the features 'Unnamed: 0', 'Flow ID', 'Timestamp', 'Source IP', 'Destination IP', 'Source Port', and 'Destination Port'. Consequently, our experiments are conducted using CIC-DDoS2017_Exp.rar [133], with the distribution of instances shown in Figure 3.5.

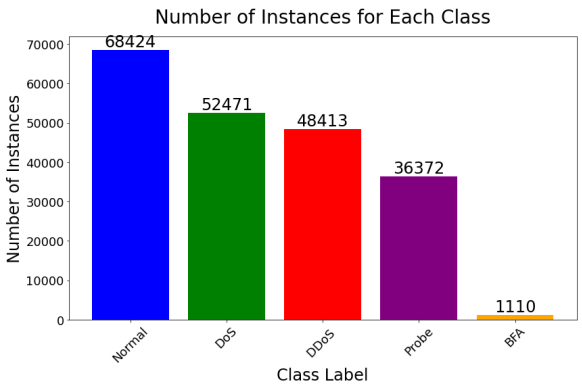


Figure 3.4: Number of instances for each class within our experimental InSDN dataset

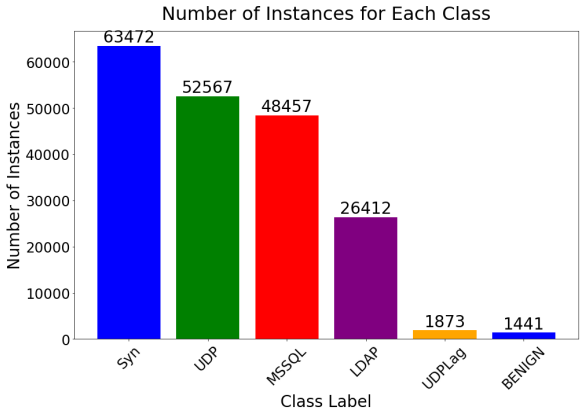


Figure 3.5: Number of instances for each class within our experimental CIC-DDoS2019 dataset

The two datasets utilized in our experiments are divided into two distinct subsets: 80% for training and 20% for testing. The testing subset evaluates the model's performance on entirely new, unseen data, offering a reliable measure of its effectiveness.

3.3.2 Methodology for the Construction and Selection of the Optimal Model

This section explains the process of developing and identifying the most effective model. Each model incorporates several trained K-Means models, along with trained Word2Vec and neural network models.

Figure 3.6 visually represents this approach. For each k value within a defined range, a model is generated, and its performance evaluated using the test dataset. The model achieving the best accuracy on the test data is selected as the optimal one. The steps for constructing a model based on this approach are outlined below:

- **Step 1:** Train separate K-Means models using the training dataset, with one model for each feature, excluding the target variable 'Label'. The total number of K-Means models corresponds to the number of features in the dataset. Each K-Means model divides the data within its respective feature into up to k clusters, focusing solely on patterns within that feature, without considering interactions between features.

  To speed up the training process, parallel processing is implemented through the joblib library's Parallel function. Following this step, each data point in a feature is assigned to a cluster using its corresponding K-Means model. The resulting cluster is converted into categorical form as 'Attribute_ClusterX', where 'X' represents the cluster ID and 'Attribute' is the feature name, with spaces replaced by underscores. This step generates a new dataset where numerical data are transformed into categorical categories.

- **Step 2:** Create a text-based feature as input for the Word2Vec model. A new column named 'text' is added to the dataset, which was generated in the previous step. In this 'text' column, each data point is a list of strings, created by converting the categorical data (excluding the label column) from each row into strings and combining them into a list. This transformation effectively converts each row into a text-based representation, which can then be processed by Word2Vec.

- **Step 3:** Train the Word2Vec model using the text feature. The Word2Vec algorithm, commonly used in natural language processing, is applied to generate dense vector representations for words. These vectors capture semantic similarities based on the context in which words appear. A vector size of 300 dimensions is chosen based on prior research demonstrating its effectiveness [134], [135]. The model uses a context window of five words on either side of a target word to learn contextual relationships. Training efficiency is enhanced by utilizing four threads for parallel processing.

- **Step 4:** Transform the text data into a numerical representation. To achieve this, extract the corresponding word embedding for each word in the text feature from the Word2Vec model's vocabulary, then compute the mean of these embeddings. If no valid word vectors are found in a text instance, append a zero vector to represent that instance.

- **Step 5:** Use the vector representations of the text data from the previous step as input to the neural network. The architecture consists of three dense layers: two hidden layers with 64 ReLU-activated units each and an output layer corresponding to the number of unique classes in the target variable. The output layer applies softmax activation to compute class probabilities. Model training utilizes the Adam optimizer and the sparse categorical cross-entropy loss function, with accuracy tracked as the evaluation metric. Since the Adam optimizer updates model parameters

using gradient calculations from training batches, there can be slight variations in model performance due to its stochastic nature. Early stopping is employed to prevent overfitting by monitoring validation performance during training, ensuring the neural network remains robust.

A detailed explanation of our proposed approach is provided in Algorithm 2, offering a thorough overview of our methodology.

Algorithm 2: Proposed Traffic Classification Approach

Input : Training set and Testing set

Output: best_KMeans_models, best_Word2Vec_model, and best_NN_model

Initialize best_accuracy $= 0$;
Initialize best_KMeans_models;
Initialize best_Word2Vec_model;
Initialize best_NN_model;
Define X as the matrix of feature values, and y as the vector of target values;
for *each k in a predefined range* **do**
    Train K-Means models KMeans_models$_k$ using Parallel Processing and assign discrete categories;
    for *each column in X_{train}* **do**
        Train a K-Means model KMeans_model$_{k,\text{column}}$ on $X_{\text{train}}[\text{column}]$;
        Append KMeans_model$_{k,\text{column}}$ to KMeans_models$_k$;
    Assign clusters to X_{train} using KMeans_models$_k$;
    Label each column's data with 'Attribute_ClusterX', where 'X' denotes the cluster ID from KMeans_models$_k$ and 'Attribute' represents the feature name with spaces replaced by underscores;
    Add a new column named 'text' to the dataset, where each entry is obtained by converting the categorical data into string format and combining them into a list;
    Train a Word2Vec model Word2Vec_model$_k$ on the `text` feature;
    for *each* `text` *entry* **do**
        Extract word vectors for valid words in the vocabulary of Word2Vec_model$_k$;
        Calculate the mean of these word vectors to obtain a single vector representation;
        Append a zero vector if no valid word vectors are found;
    Define and compile the Neural Network model;
    Train the Neural Network Model NN_model$_k$ using the vector representations of the text feature;
    Evaluate the trained model (KMeans_models_k+Word2Vec_model_k+NN_model_k) on Testing Set and compute testing accuracy accuracy$_k$;
    if *accuracy$_k$ > best_accuracy* **then**
        Update best_accuracy to accuracy$_k$;
        Update best_KMeans_models to KMeans_models$_k$;
        Update best_Word2Vec_model to Word2Vec_model$_k$;
        Update best_NN_model to NN_model$_k$;
return best_KMeans_models, best_Word2Vec_model, and best_NN_model;

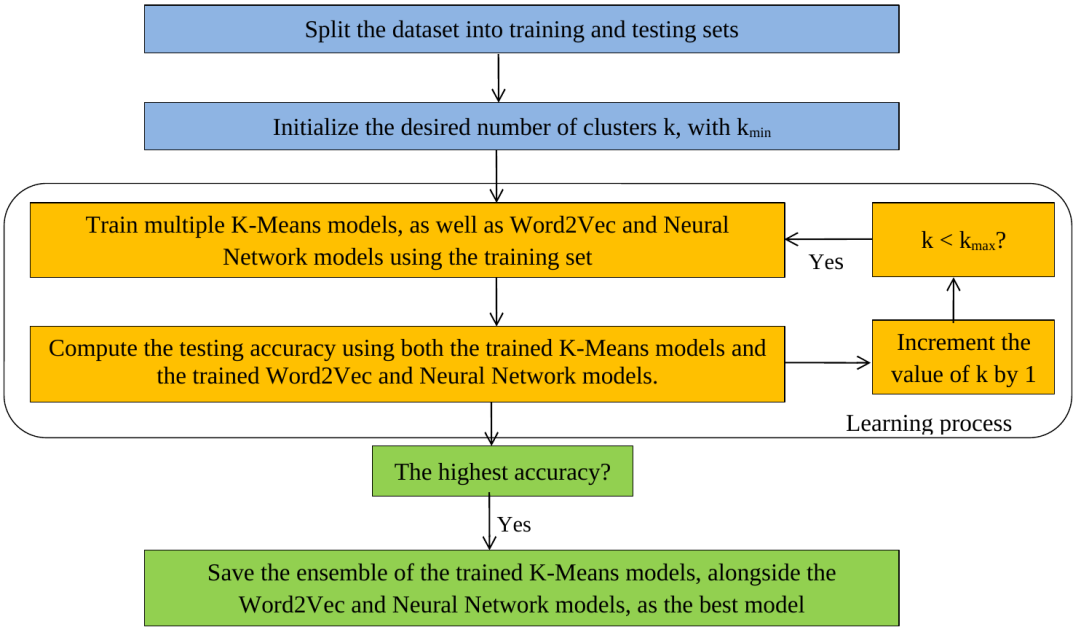


Figure 3.6: Flow diagram of the proposed approach for traffic classification

3.4 Results and Discussion

In this section, we describe the approach used to evaluate the effectiveness of the proposed method. The experimentation procedures and techniques are outlined, followed by the presentation of the results, which offer insights into the method's effectiveness and capabilities.

3.4.1 Evaluation Process and Criteria

The evaluation of our approach involves three separate experiments, each repeated 20 to 25 times. In each iteration, the desired number of clusters (k) is adjusted, and the model is trained using the chosen k value. However, the neural network model remains constant across all iterations.

For all experiments, both models are trained on the same training dataset. The neural network model uses the same classification architecture as the proposed approach. Performance is evaluated using the same test set, with standard metrics such as accuracy, precision, recall, and F1-score. The first experiment, carried out on the InSDN dataset, is conducted without feature selection (FS) to examine if K-Means clustering and Word2Vec feature extraction improve the performance of the neural network model. The second experiment, also using the InSDN dataset, is similar but utilizes only the relevant features (listed in Table 3.1). SelectKBest from the scikit-learn library is used to select the optimal number of features. The third experiment, conducted on the CIC-DDoS2019 dataset, evaluates the method's applicability beyond SDN environments and is performed without feature selection.

Table 3.1: Extracted subset features from our experimental InSDN dataset

No.	Feature	No.	Feature	No.	Feature
1	Protocol	15	Fwd IAT Max	29	SYN Flag Cnt
2	Flow Duration	16	Bwd IAT Tot	30	PSH Flag Cnt
3	Fwd Pkt Len Max	17	Bwd IAT Mean	31	ACK Flag Cnt
4	Fwd Pkt Len Min	18	Bwd IAT Std	32	URG Flag Cnt
5	Bwd Pkt Len Max	19	Bwd IAT Max	33	Down/Up Ratio
6	Bwd Pkt Len Min	20	Bwd PSH Flags	34	Pkt Size Avg
7	Bwd Pkt Len Mean	21	Bwd URG Flags	35	Fwd Seg Size Avg
8	Bwd Pkt Len Std	22	Fwd Pkts/s	36	Bwd Seg Size Avg
9	Flow Pkts/s	23	Bwd Pkts/s	37	Init Bwd Win Byts
10	Flow IAT Std	24	Pkt Len Min	38	Idle Mean
11	Flow IAT Max	25	Pkt Len Max	39	Idle Std
12	Fwd IAT Tot	26	Pkt Len Mean	40	Idle Max
13	Fwd IAT Mean	27	Pkt Len Std	41	Idle Min
14	Fwd IAT Std	28	FIN Flag Cnt		

3.4.2 Experimental Results

This section outlines the findings from the experiments using the proposed method. Initially, we compare the accuracy of the NN model and our model by adjusting the value of k. Next, we evaluate the top-performing model from our approach against the best NN model, assessing them based on precision, recall, and F1-score. We also examine the training time for both models using the InSDN dataset. The progression of accuracy through different iterations, each with a distinct k value, is visually represented in Figures 3.7, 3.9, and 3.11. Detailed accuracy values for each iteration are available in Table 3.2. Comparisons of classification reports are provided in Figures 3.8, 3.10, and 3.12, while Figures 3.13 and 3.14 show the training time comparison.

A. Evaluation on Our Experimental InSDN Dataset Without Feature Selection

Figure 3.7 illustrates that after reaching k=40, two key points stand out: our model achieves higher accuracy than the neural network model and continues to improve, reaching its highest accuracy at k=115. From k=65 onward, accuracy stabilizes, showing consistent performance across different values of k. As a result, we focus our comparison starting from iteration 8, which corresponds to k=65.

Table 3.2 shows that our model achieves a maximum accuracy of 99.97%, which is 0.20% higher than the 99.77% maximum accuracy of the NN model. The mean accuracy of our model is 99.95%, while the NN model has a mean accuracy of 99.70%. Furthermore, our model exhibits a smaller variance of 0.0003, indicating greater stability, while the NN model shows a variance of 0.0067, suggesting higher variability. These results highlight our model's superior accuracy and more consistent performance across different cluster values compared to the NN model.

The classification results for our model at k=115, shown in Figure 3.8, indicate outstanding performance in precision, recall, and F1-score for all categories. Specifically, our model achieves perfect precision (1) for detecting DoS attacks, ensuring accurate identification. It also scores 1 for precision, recall, and F1-score for Normal traffic, highlighting its ability to correctly identify normal network activity. In comparison, the neural network model performs similarly but with slightly lower precision for

DoS attacks and reduced precision (0.94) and recall (0.84) for the BFA category. This suggests that the NN model faces some difficulties in accurately identifying BFA attacks. These results emphasize the reliability and effectiveness of our model in correctly classifying both attack types and normal traffic.

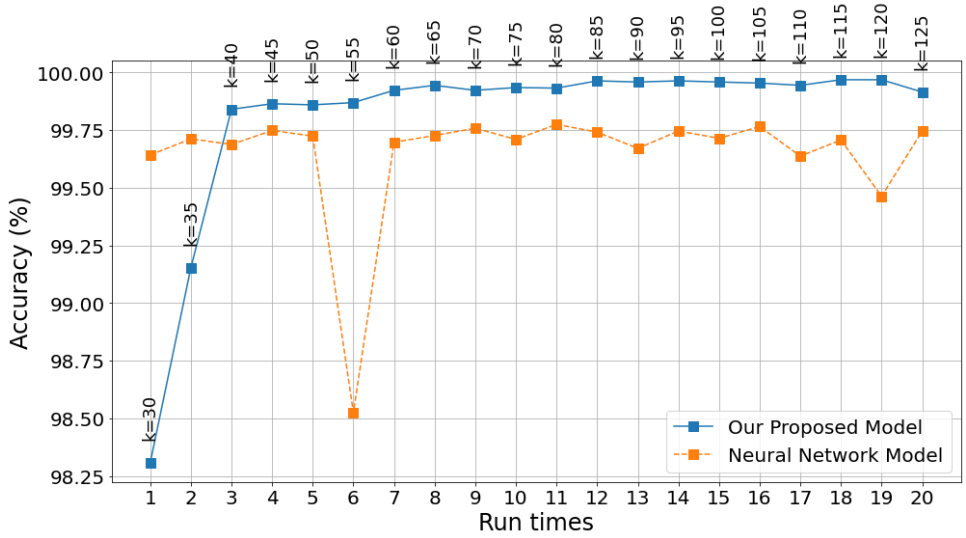


Figure 3.7: Accuracy within our InSDN dataset (No Feature Selection)

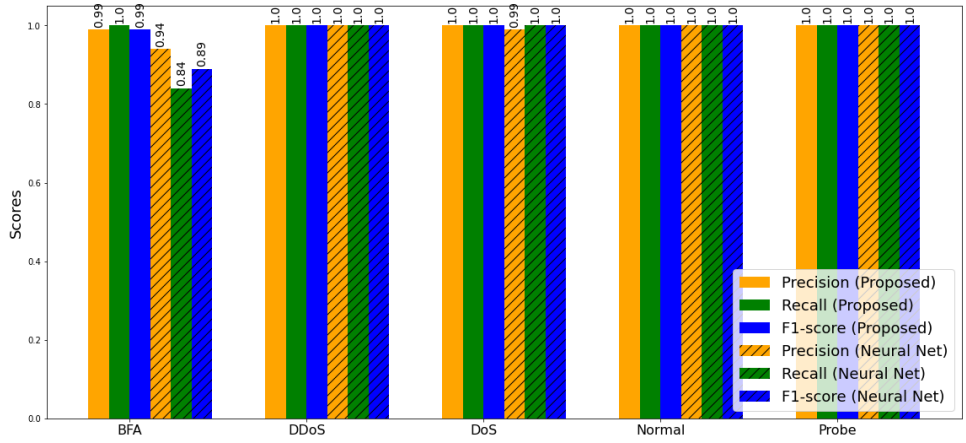


Figure 3.8: Classification reports comparison within our InSDN dataset (No Feature Selection)

Table 3.2: Accuracy of our model and NN model over iterations (%)

Iteration	**InSDN, No FS**		**InSDN with FS**		**CIC-DDoS2019, No FS**	
	Our model	**NN model**	**Our model**	**NN model**	**Our model**	**NN model**
1	98.31	99.64	95.08	97.99	97.50	97.81
2	99.15	99.71	97.57	97.94	97.64	97.32
3	99.84	99.69	97.81	98.00	97.57	98.00
4	99.86	99.75	98.09	97.97	98.27	97.49

Iteration	InSDN, No FS		InSDN with FS		CIC-DDoS2019, No FS	
	Our model	NN model	Our model	NN model	Our model	NN model
5	99.86	99.72	98.23	97.74	98.19	97.25
6	99.87	98.53	98.29	97.84	98.29	97.77
7	99.92	99.70	98.29	97.13	98.24	97.59
8	99.94	99.73	98.58	98.03	98.28	97.94
9	99.92	99.76	98.79	98.05	98.53	97.86
10	99.93	99.71	98.74	97.76	98.24	97.91
11	99.93	99.78	98.81	98.02	98.50	97.93
12	99.96	99.74	98.92	98.00	98.47	97.25
13	99.96	99.67	98.77	97.99	98.36	97.68
14	99.96	99.75	98.80	97.50	98.54	97.76
15	99.96	99.71	98.86	97.89	98.59	97.90
16	99.95	99.77	98.82	97.97	98.58	98.01
17	99.94	99.64	98.81	98.01	98.42	97.74
18	99.97	99.71	98.90	97.99	98.43	97.94
19	99.97	99.46	98.80	97.95	98.49	97.90
20	99.92	99.75	98.85	97.90	98.59	97.85
21					98.49	97.69
22					98.61	97.37
23					98.65	97.24
24					98.47	97.76
25					98.57	97.53

B. Evaluation on Our Experimental InSDN Dataset with Feature Selection

Figure 3.9 shows that our model outperforms the neural network model in terms of accuracy starting from around k=20. After k=45, the accuracy stabilizes with only small variations, indicating no substantial improvement. As a result, we focus our analysis from run time 9, which corresponds to k=45.

From Table 3.2, we see that our model reaches a peak accuracy of 98.92%, surpassing the NN model's best accuracy by 0.77%. Our model maintains a mean accuracy of about 98.82% with a low variance of 0.0027, while the NN model has a mean of 97.92% and a much higher variance of 0.0237. This highlights that our model not only achieves a higher peak accuracy but also shows more stability and consistency in performance.

Figure 3.10 further demonstrates that our model with k=60 delivers better precision for all classes compared to the neural network model. Precision scores for our model range from 0.95 to 1.00, demonstrating its strong performance, even for the BFA class with fewer instances. Additionally, our model consistently outperforms the NN model in recall, with values ranging from 0.96 to 1.00. The recall for the BFA class is particularly remarkable, with our model achieving a score of 1, while the NN model only reaches 0.5. These improvements in precision and recall are reflected in the F1-scores, where our

model consistently achieves higher values, ranging from 0.97 to 1.00. These results underscore the effectiveness of our model in accurately classifying different types of network traffic, which is critical for cybersecurity applications in real-world settings.

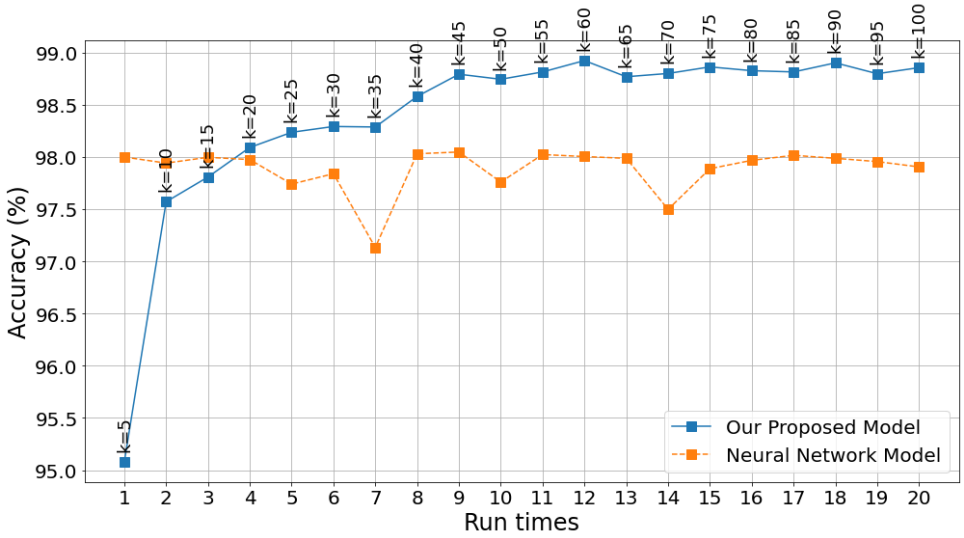


Figure 3.9: Accuracy within our InSDN dataset (with Feature Selection)

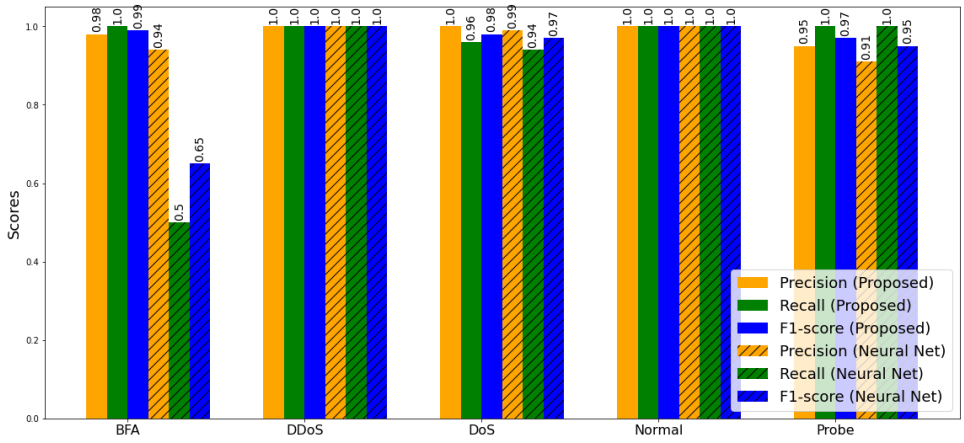


Figure 3.10: Classification reports comparison within our InSDN dataset (with Feature Selection)

C. Evaluation on Our Experimental CIC-DDoS2019 Dataset

The findings in Figure 3.11 emphasize the strong performance of our proposed model on the CIC-DDoS2019 dataset. During experiments with different cluster numbers, our model began outperforming the NN model in accuracy at around k=90. From approximately k=190 (runtime 14), accuracy remained consistently above 98.4%, demonstrating greater stability. Therefore, we will focus our comparison from iteration 14 onward.

According to Table 3.2, our model achieves an average accuracy of 98.54% and a highest accuracy of 98.65%, surpassing the NN model's peak accuracy of 98.01% by 0.64%. Additionally, our model has

a variance of 0.0056, whereas the NN model's variance is 0.0556, indicating greater fluctuations. These results highlight both the superior accuracy and the enhanced stability of our model compared to the NN model.

Figure 3.12 compares the classification reports of our model with k=280 and the NN model, revealing noticeable differences in performance metrics. Our model consistently outperforms the NN model in precision, recall, and F1-score across several classes, including BENIGN, LDAP, MSSQL, Syn, and UDP. Even for the UDPLag class, which has fewer instances, our model maintains superior precision, recall, and F1-score. Overall, our model shows stronger performance across all classes, confirming its effectiveness in classifying network traffic.

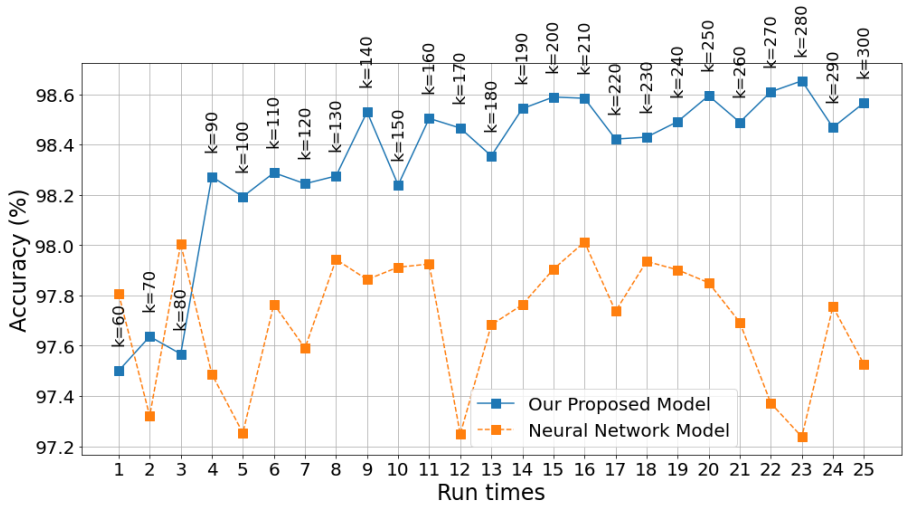


Figure 3.11: Accuracy within our CIC-DDoS2019 dataset (No Feature Selection)

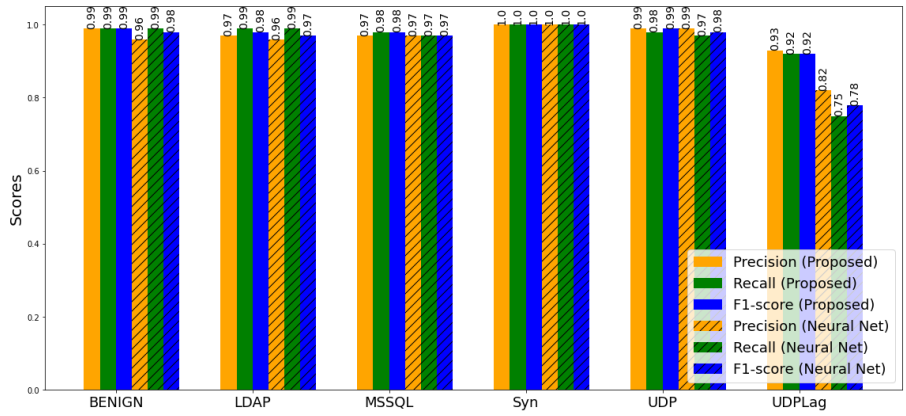


Figure 3.12: Classification reports comparison within our CIC-DDoS2019 dataset (No Feature Selection)

D. Computational Overhead on Our Experimental InSDN Dataset

The proposed approach follows a multi-step process that incurs high computational overhead, primarily due to the integration of Word2Vec and a neural network. Training the Word2Vec model, which is based

on a neural network, alongside a neural network classifier requires significant computational resources.

As shown in Figures 3.13 and 3.14, training times for the proposed model vary significantly and tend to increase with the number of clusters. Without feature selection, training times range from approximately 328.79 to 841.08 seconds, highlighting the high computational cost. In contrast, applying feature selection significantly reduces training times, ranging from 158.40 to 449.53 seconds. Feature selection reduces computational overhead. By selecting only the most relevant features, the model processes less data, leading to faster training times and lower memory usage.

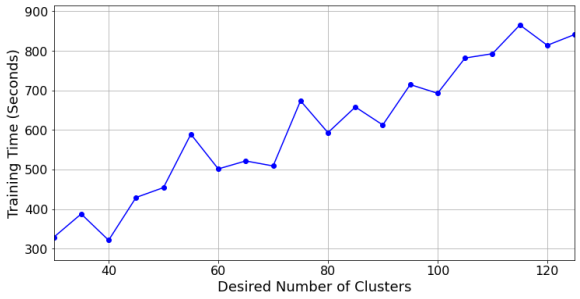


Figure 3.13: Training time comparaison without feature selection

Figure 3.14: Training time comparaison with feature selection

3.4.3 Comparison

Our method aims to improve the accuracy of traffic flow classification. To evaluate its performance, we compare it with other methods using accuracy as the benchmark. As shown in Table 3.3, our model achieves 99.97% accuracy on the InSDN dataset, outperforming several well-known methods. For example, [124] reports 98.85% accuracy using K-Means and KNN on simulated data, while our model performs better.

Additionally, the method in [136] reaches 96% accuracy, and the GRU model in [137] achieves 99.65%, but our model performs better than both. The SAE-MLP classifier in [138], with 99.75% accuracy, and the C-Support Vector Classification method in [120] with 99.67%, are also outperformed by our model.

Moreover, [121] reports accuracies of 90% and 80.7% for GRU-RNN and DNN on the NSL-KDD dataset, respectively, while our model surpasses these on both the InSDN and CIC-DDoS2017 datasets. Similarly, [122] reports 98.81% accuracy using MLP and Feedforward ANN, and [123] achieves 99% accuracy using RF, but our model achieves higher accuracy in comparison.

Table 3.3: Comparison of our proposed model (K-Means+Word2Vec+NN) with existing approaches

Reference	Dataset	Purpose of Study	ML Techniques	Accuracy
[124]	Simulated data	DDoS Detection	Combined K-Means and KNN	98.85%
[120]	Simulated data	Anomaly Detection	CSVC (C-Support Vector Classification)	99.67%

Continued on next page

Reference	Dataset	Purpose of Study	ML Techniques	Accuracy
[121]	NSL-KDD	Network Intrusion Detection	DNN and GRU-RNN	GRU-RNN achieves 90%, surpassing DNN's 80.7%
[122]	Self-generated dataset	Network Traffic Classification	MLP and Feedforward (FF) ANN	98.81%
[123]	InSDN dataset	Network Traffic Classification	RF, KNN, and DT	RF classifier achieved 99%
[136]	Real-world traffic dataset	Network Traffic Classification	Deep learning classifier	96%
[137]	InSDN and ISCX-VPN-NoVPN	Network Traffic Classification	GRU	99.65%
[138]	Provided by India Project Mentor	DDoS Detection	SAE-MLP classifier	99.75%
Proposed model	InSDN and CIC-DDoS2019	Network Traffic Classification	Combination of K-Means, Word2Vec, and Neural Network	99.97% (InSDN, no FS), 98.92% (InSDN, FS), 98.65% (CIC-DDoS2019, no FS)

3.4.4 Analysis and Discussion

The experimental results demonstrate that our method outperforms traditional neural network models in terms of both accuracy and consistency across various datasets. On the InSDN dataset, our approach achieves 99.97% accuracy without feature selection when the desired number of clusters is 115, outperforming other established methods listed in Table 3.3. It continues to maintain strong accuracy with feature selection, particularly when k exceeds 45. On the CIC-DDoS2019 dataset, our method delivers consistent performance, maintaining an average accuracy of approximately 98.54% from around k = 190 onwards. This stability, along with the high precision, recall, and F1-scores across different traffic types, highlights the method's effectiveness in classifying network traffic, even with smaller sample sizes.

The process of training multiple K-Means models, along with Word2Vec and NN models, leads to considerable computational overhead, which increases with the desired number of clusters. Feature selection helps reduce this overhead and improves efficiency. Additionally, cloud computing and NN pruning and quantization can further mitigate computational costs, enhancing scalability and making the method more practical for real-world applications.

The techniques developed in this research offer valuable applications in fields that require high classification accuracy, particularly when handling imbalanced datasets. This includes areas such as medical diagnostics and fraud detection, where precise classification is crucial despite uneven data distributions.

3.5 Conclusion

This study highlights the significance of feature-based methods in strengthening SDN security through advanced clustering and feature extraction techniques. Our model, which integrates multiple K-Means models for clustering, Word2Vec for feature extraction, and a neural network for traffic classification, consistently delivers high accuracy and stability across various traffic types, including those with limited sample sizes. These results emphasize the potential of our method to identify and classify complex patterns within SDN environments, surpassing traditional neural network model and other existing methods. However, the computational demands of using multiple K-Means models and Word2Vec underscore the importance of efficient parallelization and robust infrastructure to ensure scalability and practical deployment in real-world settings.

We explored the NLP algorithm Word2Vec for feature extraction. Our next contribution aims to explore computer vision algorithms for feature extraction from datasets. To achieve this, we introduce a unique method for converting raw data into images and then apply either the ORB or SIFT algorithm for feature extraction.

Chapter 4

Evaluating ORB and SIFT with Neural Network as Alternatives to CNN for Traffic Classification in SDN Environments

In the previous chapters, we introduced a machine learning-based approach for flooding DDoS detection using multiple K-Means clustering and Naïve Bayes, leveraging clustering to improve attack detection. Building on this, we proposed a feature-engineering-based method that combines K-Means, Word2Vec, and a neural network, extending detection capabilities to a broader range of attacks by incorporating semantic relationships in traffic features.

This chapter further expands on these efforts by exploring an image-based classification approach for SDN traffic analysis. We introduce a novel method that converts flow data into images, enabling feature extraction using either ORB or SIFT, followed by classification with a neural network. To evaluate its effectiveness, we compare this method against a CNN model, demonstrating its potential as an alternative that achieves a good balance between accuracy and computational overhead.

CNNs are widely recognized for their ability to process complex spatial patterns, making them highly effective for image classification tasks [139]. Their automatic feature learning capability eliminates the need for manual feature engineering while maintaining high accuracy and robustness through translation invariance. However, despite these advantages, CNNs are computationally demanding [140], which can be a drawback in resource-constrained environments.

While recurrent architectures such as RNNs, LSTMs, and GRUs excel at capturing temporal dependencies in sequential data, they may not be the most suitable for SDN traffic classification. Spatial relationships derived from flow data are crucial for accurate classification, making CNNs a more effective choice when data is transformed into an image-like structure. Additionally, RNN-based models tend to incur higher computational overhead and longer training times [141, 142], making real-time classification more challenging in SDN environments. In comparison, CNNs not only achieve competitive accuracy but also outperform DNNs, RNNs, LSTMs, and GRUs, as reported in [143].

To assess the effectiveness of our ORB and SIFT-based models, we conduct a comparative analysis using the InSDN dataset. The results indicate that the ORB model outperforms the SIFT model and remains highly competitive with the CNN model. Notably, while the ORB model achieves an accuracy only 1.99% lower than the CNN model when flow data is converted into a bar chart, it offers significantly lower computational overhead.

Additionally, we evaluate our method using three different flow-to-image conversion techniques: our unique bar chart-based approach and two well-established methods, IGTD [144] with Euclidean distance and IGTD with Manhattan distance. The results demonstrate that models perform particularly well when flow data is converted into bar charts. Finally, we assess the classification capability of the ORB model using flow-to-bar chart conversion on the CIC-DDoS2019 dataset, achieving an accuracy of 99.86% in distinguishing between various types of DDoS attacks and normal traffic. These findings highlight the model's effectiveness and its potential applicability beyond SDN environments.

The next section provides an overview of SIFT, ORB, and CNN models, detailing their key components, advantages, and limitations.

4.1 Overview of SIFT, ORB and CNN

4.1.1 Scale-Invariant Feature Transform

The SIFT algorithm [145] is a widely used computer vision algorithm for detecting and describing local features in images. It is highly robust to variations in scale and rotation and exhibits partial invariance to changes in illumination and affine transformations. The SIFT algorithm consists of four main phases, as illustrated in Figure 4.1. The key phases of SIFT are as follows:

Scale-Space Extrema Detection: Key points that are invariant to scale and orientation changes are identified by constructing a scale-space representation. This is achieved by applying Gaussian blurring at different scales and forming an image pyramid with progressively lower resolutions. The Difference of Gaussians (DoG) is computed by subtracting adjacent Gaussian-blurred images at each level. Key points are then detected as local maxima and minima in the DoG images by comparing each pixel to its neighbors across scales.

Keypoint Localization: Once potential keypoints are identified, they are refined to improve stability and precision. A three-dimensional quadratic function interpolates their exact location. Low-contrast keypoints and those situated along edges are removed to enhance robustness.

Orientation Assignment: Each keypoint is assigned one or more orientations based on local image gradients to ensure rotation invariance. A histogram of gradient directions within the keypoint's neighborhood is computed, and the dominant peak(s) in the histogram determine its orientation(s). This orientation assignment step is crucial because the keypoint descriptor is then constructed relative to this orientation, ensuring that the descriptor is rotation-invariant.

Keypoint Descriptor Construction: Once the orientation(s) are assigned to the keypoint in the previous step, a region around each keypoint is divided into 4×4 subregions. In each subregion, an 8-bin gradient histogram is calculated. These histograms represent the distribution of gradients in the

region and are aligned relative to the keypoint's orientation. The histograms are then normalized for illumination invariance and concatenated to form a descriptor vector that remains invariant to scale, orientation, and illumination changes. The final descriptor consists of 128 values, ensuring robustness for matching tasks.

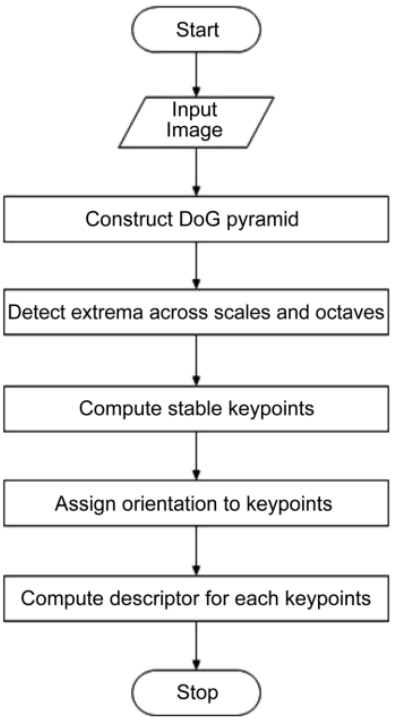


Figure 4.1: SIFT algorithm flowchart

While SIFT is a powerful feature detection and description algorithm, it has several limitations that impact its performance in real-world applications. The algorithm is computationally expensive, making it less suitable for real-time applications, especially for high-resolution images. Its 128-dimensional descriptor requires significant memory, which can slow down feature matching in large datasets. Although SIFT is resilient to minor noise variations, it may produce unstable key points in highly noisy environments. Additionally, while SIFT exhibits partial invariance to affine transformations, its performance degrades under extreme perspective distortions.

4.1.2 Oriented FAST and Rotated BRIEF

The ORB algorithm [146] is an efficient, rotation-invariant method for keypoint detection and feature description, widely used in real-time applications. ORB builds on the FAST (Features from Accelerated Segment Test) [147] detector for keypoint identification and the BRIEF (Binary Robust Independent Elementary Features) [148] descriptor for feature representation. It enhances both components by introducing rotation invariance and multi-scale analysis, improving robustness to variations in scale, illumination, and orientation.

The ORB algorithm consists of several key components that work together to achieve efficient feature detection and description:

- **Multi-Scale Representation and Keypoint Detection:** ORB employs an image pyramid, where each level is a progressively downsampled version of the image, ensuring keypoint detection across multiple scales. The FAST corner detector identifies keypoints by evaluating intensity variations within a Bresenham circle of radius 3 pixels around a central pixel. A pixel is classified as a corner if at least *n* contiguous pixels in the circle are significantly brighter or darker than the center pixel, with ORB typically using $n = 9$.

- **Keypoint Orientation Assignment:** Since FAST lacks orientation awareness, ORB assigns keypoint orientation using an intensity-weighted centroid method. The centroid coordinates (c_x, c_y) are computed as:

$$c_x = \frac{m_{10}}{m_{00}}, \quad c_y = \frac{m_{01}}{m_{00}} \tag{4.1}$$

where the image moments are defined as:

$$m_{00} = \sum_{(x,y)\in W} I(x,y), \quad m_{10} = \sum_{(x,y)\in W} x \cdot I(x,y), \quad m_{01} = \sum_{(x,y)\in W} y \cdot I(x,y) \tag{4.2}$$

Here, $I(x, y)$ represents the grayscale intensity of the pixel at coordinates (x, y), typically ranging from 0 (black) to 255 (white). The summation is performed over a window W around the keypoint, which is typically a square region centered at the keypoint.

The keypoint orientation θ is then computed as:

$$\theta = \arctan 2(m_{01}, m_{10}) \tag{4.3}$$

This ensures that keypoints maintain a consistent orientation across rotated images.

- **Descriptor Extraction Using Rotation-Invariant BRIEF (rBRIEF):** The BRIEF descriptor generates a binary feature descriptor by comparing intensity values of sampled pixel pairs. Since standard BRIEF is sensitive to rotation, ORB introduces rBRIEF, aligning the sampling pattern with the keypoint's orientation θ:

$$[x_r, y_r] = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} [x, y] \tag{4.4}$$

For each keypoint, 256 intensity comparisons are performed on pixel pairs, generating a 256-bit binary descriptor. This descriptor is then stored as a 32-byte vector, with each byte representing an 8-bit value.

$$d_i = \begin{cases} 1, & \text{if } I(x_r^i, y_r^i) > I(x_r^j, y_r^j) \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

where (x_r^i, y_r^i) and (x_r^j, y_r^j) are sampled points in the rotated BRIEF pattern.

ORB offers several advantages that make it suitable for real-time applications. It is computationally efficient, performing faster than SIFT while maintaining robust keypoint detection and description. Its rotation and scale invariance, achieved through keypoint orientation assignment and multi-scale representation, enhances its performance under varying conditions. Additionally, ORB uses binary descriptors, which require less storage and enable faster matching, making it well-suited for resource-constrained environments.

Despite its benefits, ORB has some limitations. It is sensitive to perspective changes, making it less effective under significant viewpoint transformations. In low-texture regions, ORB detects fewer keypoints, reducing its ability to capture distinctive features in smooth areas. Furthermore, since ORB relies on local features, it lacks global context, which can be a drawback when recognizing larger structural patterns.

4.1.3 Convolutional Neural Networks

CNNs are a class of deep learning models specifically designed for processing structured grid data, such as images. They are inspired by the human visual system and are particularly effective in tasks like image and video recognition, object detection, and segmentation.

A. Architecture

A typical CNN architecture consists of several key components. Figure 4.2 illustrates the typical layers and connections in a CNN, offering a clear overview of its structure. To further understand how CNNs process data, we delve into the primary components that constitute their architecture:

- **Convolutional Layers**: These layers apply convolutional operations to the input data, using filters (also known as kernels) to detect local patterns such as edges, textures, and shapes. Each filter is convolved across the input image to produce a feature map, highlighting the presence of specific features in different regions of the image.
- **Activation Functions**: After convolution, the feature maps are passed through activation functions like Rectified Linear Unit (ReLU) to introduce non-linearity, enabling the network to learn complex patterns.
- **Pooling Layers**: Pooling operations, such as max pooling or average pooling, are used to reduce the spatial dimensions of the feature maps. This downsampling process decreases the computational load and helps in achieving spatial invariance, making the network more robust to translations and distortions in the input data.
- **Fully Connected Layers**: Towards the end of the network, fully connected layers are used to integrate the features extracted by the convolutional and pooling layers. These layers are responsible for making final predictions or classifications based on the learned features.

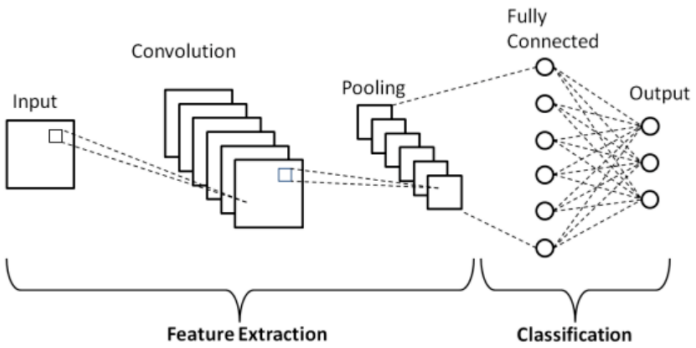


Figure 4.2: Illustration of a CNN architecture

B. Advantages

- **Automatic Feature Extraction**: CNNs can automatically learn relevant features from raw input data, eliminating the need for manual feature engineering.
- **High Accuracy**: CNNs have demonstrated state-of-the-art performance in various image and video recognition tasks.
- **Robustness to Noise**: CNNs are robust to noise and distortion in the input data, making them highly effective in real-world applications.

C. Limitations

- **High Computational Requirements**: Training CNNs, especially deep architectures, requires significant computational resources and time.
- **Need for Large Amounts of Labeled Data**: CNNs typically require large datasets for effective training, which can be a limitation in scenarios where labeled data is scarce.
- **Interpretability Challenges**: The complex nature of CNNs can make it difficult to interpret how they arrive at specific decisions, posing challenges in applications where understanding the decision-making process is crucial.

4.2 Experimental Datasets and Proposed Methodology

In this section, we introduce the experimental datasets and outline our proposed approach for evaluating three algorithms: CNN, SIFT, and ORB for traffic classification in an SDN environment. The models are implemented and evaluated using Python.

4.2.1 Experiment Datasets

The study evaluates our traffic classification method for SDN using the InSDN dataset, creating an experimental dataset [149] based on the instance distribution illustrated in Figure 4.3. To further assess

the proposed model, we employ the CIC-DDoS2019 dataset and derive an experimental dataset [150] following the instance distribution shown in Figure 4.4.

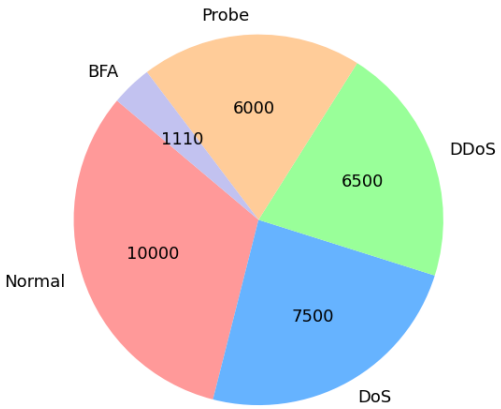


Figure 4.3: Class instance distribution in the InSDN experimental dataset for traffic classification

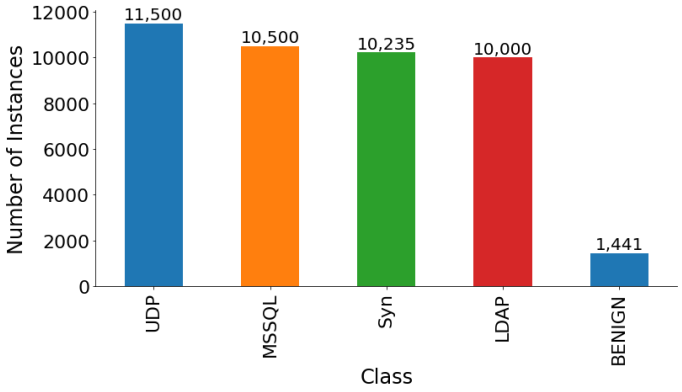


Figure 4.4: Class instance distribution within our experimental CIC-DDoS2019 dataset for traffic classification

4.2.2 Proposed Approach

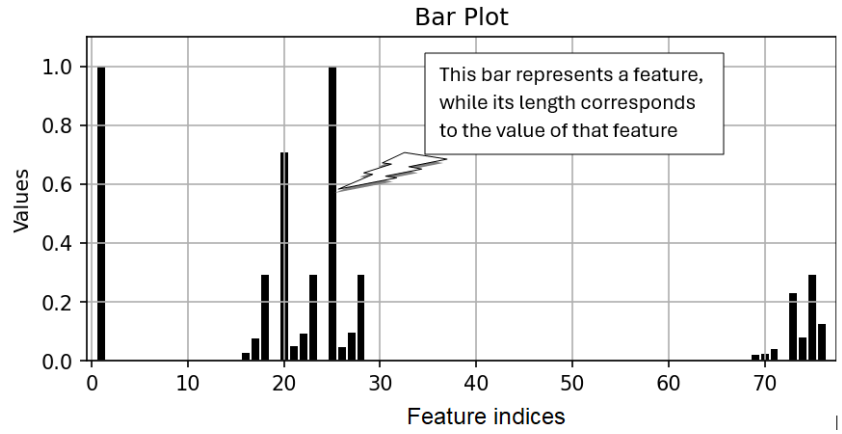
The construction process of the ORB and SIFT models consists of the following steps:

Step 1: Image Dataset Generation

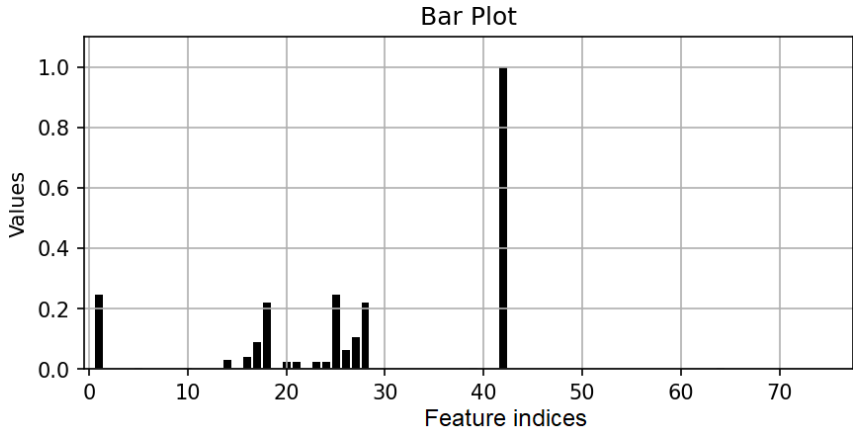
The process of image dataset generation starts by identifying unique class labels, such as 'Normal' and 'DDoS,' and creating directories with the same names as those unique classes to organize the images by class. Then, an image dataset is created by converting each row (flow) in the CSV file into an image and saving it to its corresponding directory. The last column (target variable) is excluded from each row before conversion to an image. Three methods are proposed to achieve this:

- Method 1: Flow-to-bar chart conversion. For each row (flow), features such as flow duration and mean inter-arrival time are normalized to a range of 0 to 1 using the MinMaxScaler. A bar chart is then generated for each row, where each bar represents a different feature, with the feature values

plotted on the y-axis and the feature indices represented on the x-axis. These charts are saved as PNG files and grouped by class. This process is repeated for every row in the dataset. Examples of images presenting traffic flows, generated using flow-to-bar chart conversion, are illustrated in Figure 4.5.



(a) Normal



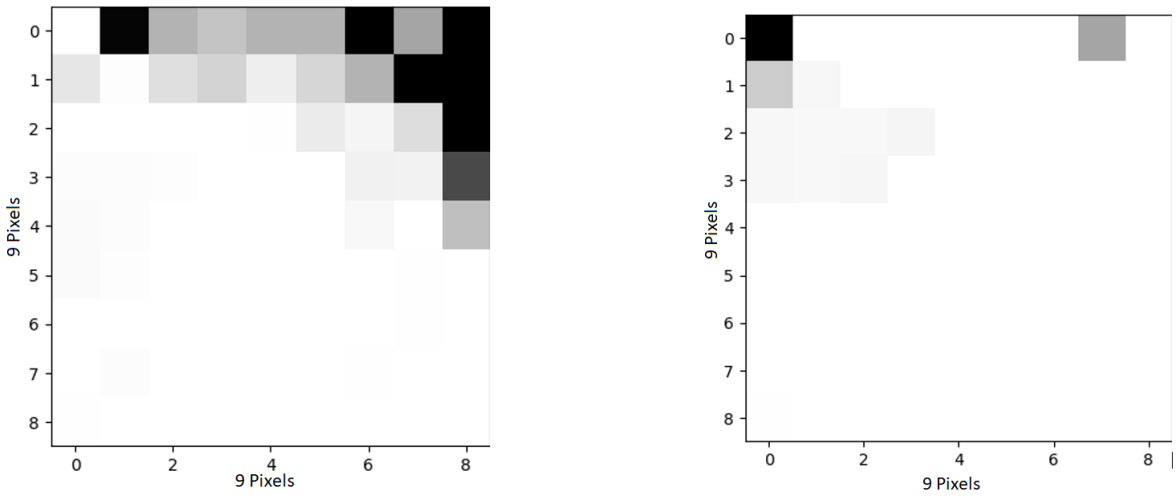
(b) BFA

Figure 4.5: Sample images generated using the flow-to-bar chart conversion technique

- Method 2: Image conversion using IGTD based on Euclidean distance. The algorithm transforms tabular data into images by mapping each feature to a corresponding pixel in a two-dimensional grid. It begins by calculating pairwise distances between features using Euclidean distance and ranking them to form a feature distance rank matrix. A similar process is applied to the pixels, where pairwise distances based on their grid positions are calculated and ranked. The algorithm then compares the feature and pixel distance rankings, aiming to minimize the difference between them. This is achieved through iterative feature swapping between pixels, ensuring similar features are placed close together, while dissimilar ones are positioned farther apart. The process continues until the feature and pixel distance rankings are closely aligned, at which point the final image is generated, with features mapped to their corresponding pixels.

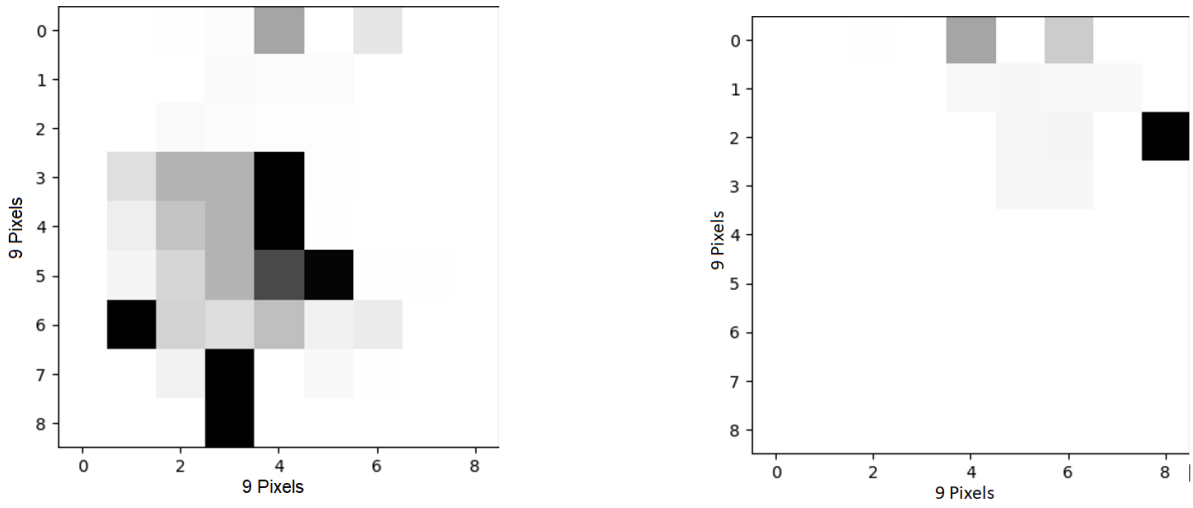
- Method 3: Similar to Method 2, but uses Manhattan distance instead of Euclidean distance. Figure 4.6 and 4.7 show images of the same flows as Figure 4.5, but generated using the IGTD algo-

rithm. In these figures, features and their values are represented by squares, with the intensities corresponding to their values.



(a) Normal (b) BFA

Figure 4.6: Sample images generated using IGTD with euclidean distance. Each square represents a feature, and its intensity corresponds to its value



(a) Normal (b) BFA

Figure 4.7: Sample images generated using IGTD with manhattan distance

Step 2: Image Dataset Split

The image dataset is split, assigning 80% to the training set and 20% to the testing set using the train_test_split function from sklearn.model_selection in Python. The test_size=0.2 parameter allocates 20% to the testing set, and the stratify=labels argument ensures the class distribution in both subsets matches the original dataset. Here, labels refer to the array of class labels, where each entry corresponds to the class label associated with its respective image.

Step 3: Feature Extraction from the Training Dataset Images Using ORB or SIFT

Feature extraction is performed on the training dataset images using either ORB or SIFT, and the extracted features are used as input for training a neural network model. For both ORB and SIFT, we set `max_keypoints` to 500, where each keypoint's descriptor is a 32-dimensional vector for ORB and a 128-dimensional vector for SIFT. Thus, the descriptor matrix has a shape of $(500, 32)$ for ORB and $(500, 128)$ for SIFT. After flattening these matrices into 1D arrays, the resulting feature vectors have a shape of $(500 \times 32) = 16,000$ for ORB and $(500 \times 128) = 64,000$ for SIFT. Consequently, the input features to the neural network are vectors of size 16,000 for ORB and 64,000 for SIFT.

Step 4: Train a Neural Network Model

A neural network with a three-layer architecture is trained for the classification task. The model includes two hidden layers, each with 64 units and ReLU activation functions, as well as an output layer with five units for classification. Dropout layers are applied after the first and second dense layers, with a dropout rate of 0.3, to mitigate overfitting. Additionally, early stopping is used as a callback during training to monitor the validation loss. If the validation loss does not improve after a specified number of epochs (patience=3), training is halted, ensuring the model reverts to its best-performing state.

Step 5: Model Evaluation

Evaluate the performance of the model on the training set images. First, features are extracted using the feature extraction model, followed by classification using the trained neural network model.

The flow diagram illustrating the construction of the ORB and SIFT models is shown in Figure 4.8, while Algorithm 3 outlines the complete approach, from feature extraction to model evaluation. The algorithm is implemented in Python.

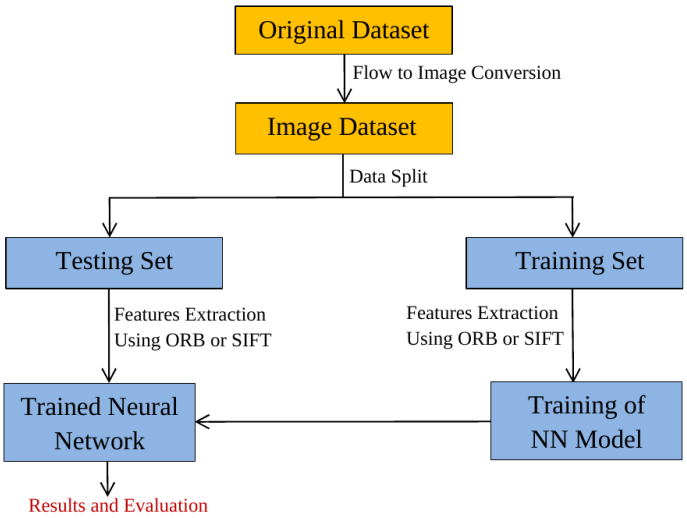


Figure 4.8: Flow diagram of the proposed method

Algorithm 3: Constructing the ORB or SIFT Model

Input : Dataset containing folders, with each folder containing images of a specific class
Output: Trained Neural Network Model

Define `max_keypoints` = 500.
Initialize list of image file paths: `image_paths` = [].
Initialize list of corresponding class labels: `labels` = [].
Step 1: Create Lists of Image File Paths and Labels
For each folder in the dataset, iterate through the files and append the image paths and corresponding labels to their respective lists.
Step 2: Convert to NumPy Arrays
Convert `image_paths` and `labels` to NumPy arrays:
Step 3: Split Dataset
Split the dataset into training and testing sets using `train_test_split` with `test_size=0.2`, `stratify=labels`, and `shuffle=True`.
Step 4: Feature Extraction using ORB or SIFT
Initialize the ORB or SIFT detector with `max_keypoints`.
for *each image in the dataset* **do**
    Submit the image and corresponding label to the thread pool for concurrent processing.
    In each thread:
    1- Read the image, convert it to grayscale, and extract ORB keypoints and descriptors.
    2- If the number of keypoints is less than `max_keypoints`, pad the descriptors with zeros.
    3- Store the valid descriptors and their corresponding labels.
After processing all images, return the list of extracted ORB features and their corresponding labels.
Step 5: Feature Standardization
Standardize the extracted ORB or SIFT features using `StandardScaler` from `sklearn`.
Step 6: Neural Network Model Construction
- Define the neural network model.
- Add layers: Dense → Dropout → Dense → Dropout → Dense (Softmax for classification).
- Compile the model with the Adam optimizer and sparse categorical cross-entropy loss.
- Train the model using the scaled features with early stopping to avoid overfitting.
Step 7: Model Evaluation
Evaluate the model on the testing set.
return Trained Neural Network Model;

4.2.3 Evaluation Process and Metrics

The performance of the CNN, SIFT, and ORB models is evaluated for each flow-to-image conversion method using our experimental InSDN dataset, based on key metrics such as accuracy, precision, recall, and F1-score.

As accuracy is a widely used metric for evaluating classification performance [151, 152], the comparison between models will be based on accuracy to identify the two models that achieved the highest accuracy and the flow-to-image conversion methods in which they excelled. Subsequently, these two models will be compared based on computational overhead. Finally, the model that optimally balances accuracy and computational overhead will be tested on our experimental CIC-DDoS2019 dataset to assess its performance beyond SDN environments, ensuring its applicability in diverse network settings.

Other techniques, such as KAZE, SURF, and AKAZE, were not included in the comparison due to their higher computational expense relative to ORB [153]. KAZE and AKAZE are computationally heavy due to their complex algorithms involving nonlinear diffusion and advanced scale-space methods, which require significant processing power and memory. KAZE employs nonlinear diffusion processes for feature detection, involving iterative computations across multiple scales. AKAZE, similar to KAZE,

incorporates optimizations for faster performance but still relies on complex calculations. SURF, though effective for feature detection, also demands considerable resources because of its fast Hessian matrix-based approach, which involves extensive matrix operations and calculations. In contrast, ORB is more efficient, utilizing simpler algorithms and compact binary descriptors that reduce both processing time and memory usage. Thus, ORB is a more practical and resource-efficient choice compared to KAZE, AKAZE, and SURF.

CNN is a widely used and highly effective deep learning architecture specifically designed for processing visual data [154]. Its ability to automatically learn features directly from input images makes it a strong benchmark in image-related tasks. Comparing ORB and SIFT models with the CNN model provides valuable insights into their performance differences. Evaluating these models together helps us understand their effectiveness in feature extraction and classification.

In this work, the CNN architecture consists of four Conv2D layers with 32, 64, and two 128 filters of size (3,3), each using ReLU activation and followed by MaxPooling2D layers with a pool size of (2,2). The input shape is (150,150,1). A Flatten layer connects to a Dense layer with 512 units (ReLU), followed by a Dropout layer (0.5), and a final Dense layer with softmax activation for classification. The model is compiled using the Adam optimizer, sparse categorical crossentropy loss, and accuracy as the evaluation metric.

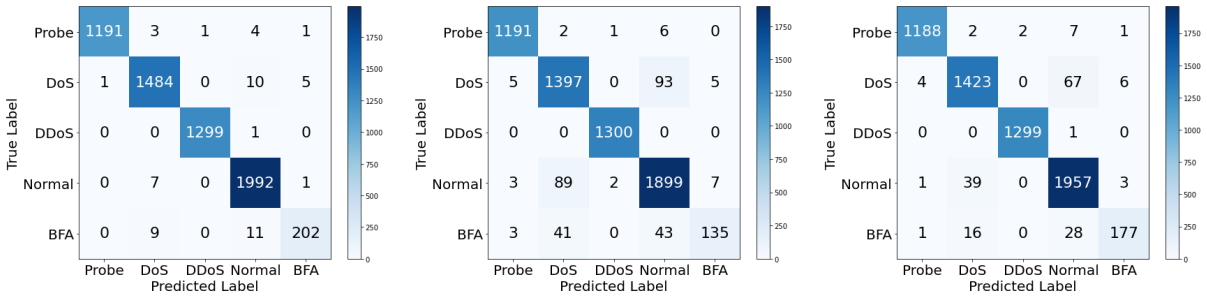
4.3 Experimental Results and Analysis

This section presents the findings of the experiments conducted using our approach, along with an analysis of the results. A comparison between CNN, ORB, and SIFT models is provided, followed by an evaluation of the optimal model on our experimental CIC-DDoS2019 dataset. Finally, the scalability of the model is analyzed, and its performance is compared with results from other studies. The discussion also highlights the novelty of the optimal model.

4.3.1 Performance Evaluation on the InSDN Dataset

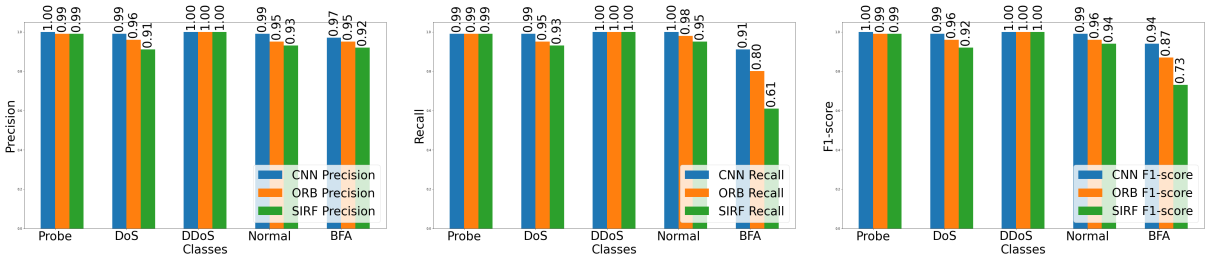
A. Result using flow-to-bar chart conversion

Based on Figure 4.9, it is evident that each model demonstrates varying degrees of performance. The CNN model exhibited exceptional accuracy, achieving an impressive score of 99.13%. Its robustness is further underscored by the classification reports comparison in Figure 4.10, which showcases consistently high precision, recall, and F1-scores across all classes. Notably, the CNN model demonstrates particularly strong performance in classifying instances of DDoS attacks, achieving perfect precision and recall. Conversely, while the SIFT model achieved a respectable accuracy of 95.18%, its precision and recall for the Normal, DoS, and BFA classes were lower than those of the CNN model. On the other hand, the ORB model attained a higher accuracy of 97.14% and demonstrated competitiveness with CNN model, despite its relatively lower precision and recall for the Normal, DoS, and BFA classes.



(a) Using CNN (b) Using SIRF (c) Using ORB

Figure 4.9: Confusion matrix using flow-to-bar chart conversion

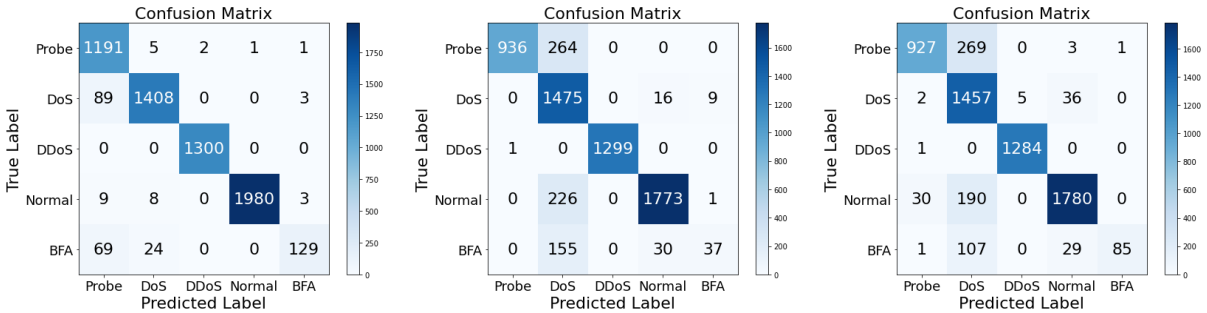


(a) Precision (b) Recall (c) F1-score

Figure 4.10: Comparison of classification reports using flow-to-bar chart conversion

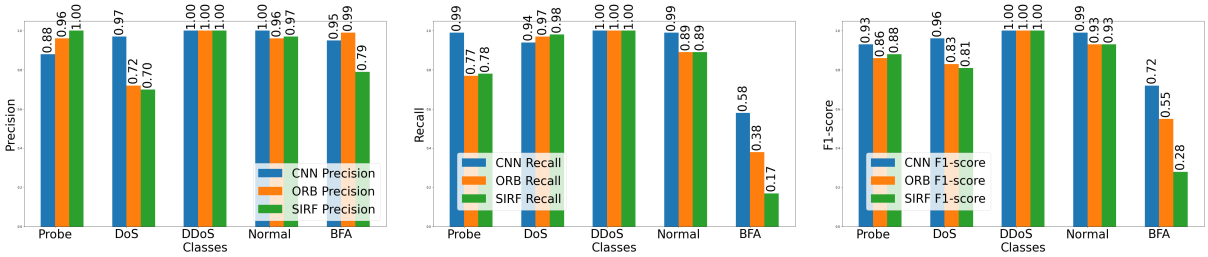
B. Result using IGTD based on euclidean distance

The results presented in Figures 4.11 and 4.12 demonstrate that the CNN model achieved the highest performance among the three models, with an accuracy of 96.56%. Although it demonstrated good precision and recall for the DDoS and Normal classes, it registered lower scores for the Probe, DoS, and BFA classes compared to the same CNN model when using flow-to-bar chart conversion. The SIFT model attained an accuracy of 88.72%, which is lower compared to the CNN model. While it performed well in the DDoS class, it faced challenges in accurately classifying instances of the Probe, DoS, BFA, and Normal classes, as reflected by the lower F1-scores for these classes. Similarly, the ORB model achieved an accuracy of 89.14%. Like the SIFT model, it encountered difficulties in accurately classifying instances of Probe, DoS, BFA, and Normal classes.



(a) Using CNN (b) Using SIFT (c) Using ORB

Figure 4.11: Confusion matrix using IGTD based on euclidean distance

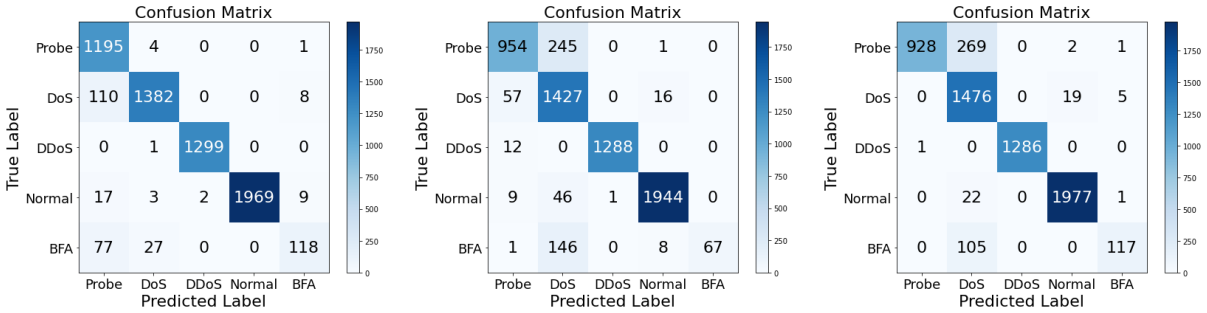


(a) Precision (b) Recall (c) F1-score

Figure 4.12: Comparison of classification reports using IGTD based on euclidean distance

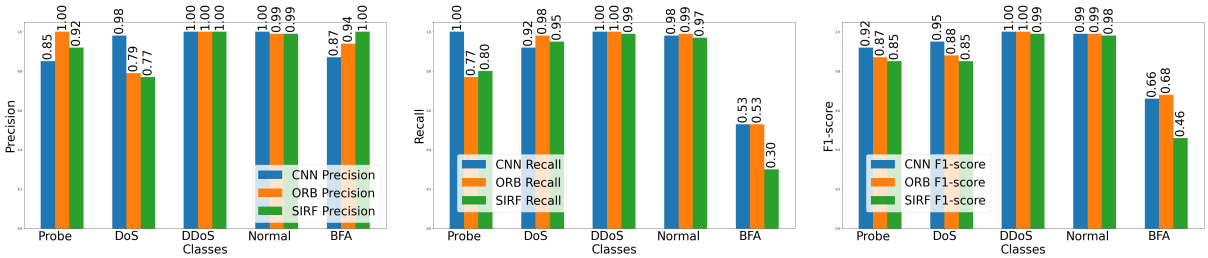
C. Result using IGTD based on manhattan distance

As illustrated in Figures 4.13 and 4.14, the CNN model outperforms the other two models, achieving the highest accuracy of 95.84%. It maintains balanced precision, recall, and F1-scores across most classes. However, it faces challenges in accurately classifying instances in the Probe and BFA classes, as indicated by lower metrics. SIFT and ORB achieve lower accuracies compared to CNN, with accuracies of 91.29% and 93.15%, respectively. They also face challenges in accurately classifying instances in the Probe, DoS, and specifically BFA classes, as indicated by their lower F1-scores.



(a) Using CNN (b) Using SIFT (c) Using ORB

Figure 4.13: Confusion matrix using IGTD based on manhattan distance



(a) Precision (b) Recall (c) F1-score

Figure 4.14: Comparison of classification reports using IGTD based on manhattan distance

The experimental results indicate that the CNN model consistently outperforms both the SIFT and ORB models across various data transformation methods, making it the best choice for traffic classification to achieve the highest accuracy, precision, recall, and F1-score. Both SIFT and ORB models exhibit their best performance with flow-to-bar chart conversion, followed by IGTD based on Manhattan

distance, and finally IGTD based on Euclidean distance. The ORB model outperforms the SIFT model. Notably, although the ORB model, using flow-to-bar chart conversion, has an accuracy 1.99% lower than the CNN model, it remains competitive due to its lower computational expense [155].

4.3.2 Computational Overhead Analysis

The computational overhead of the CNN and ORB models, both using flow-to-bar chart conversion, is compared based on key factors such as total trainable parameters, training time, testing time, average memory usage, and average CPU usage. For the CNN model, training time includes image preprocessing and CNN training, while for the ORB model, it encompasses feature extraction and neural network training. Similarly, testing time for the CNN model involves image preprocessing and testing, whereas for the ORB model, it consists of feature extraction and neural network testing.

A. Total Trainable Parameters

The total number of trainable parameters was determined using the model.summary() function in Python. As shown in Table 4.1, the ORB model has 1,028,549 trainable parameters, all coming from the neural network, as the ORB feature extraction process does not involve any trainable parameters. In contrast, the CNN model has 3,454,597 trainable parameters, with 3,213,341 parameters from the fully connected layers and 240,256 parameters from the convolutional layers. These values highlight the significant difference in model complexity, with the CNN model having a considerably higher number of parameters, reflecting its deeper architecture and greater computational demands.

Table 4.1: Total trainable parameters of ORB and CNN models

Model	Architecture	Total trainable parameters
ORB Model	- 500 keypoints per image - Descriptor size: 32 bytes per keypoint - NN with 3 layers: two hidden layers, each with 64 units and dropout rate of 0.3, and an output layer containing 5 units	1,028,549 parameters are all from the neural network; ORB feature extraction doesn't have trainable parameters.
CNN Model	- 4 Conv2D layers with 32, 64, 128, 128 filters, kernel size (3,3), 'relu' activation - MaxPooling2D layers with pool size (2,2) - Dense layer with 512 units and 'relu', followed by a Dropout of 0.5 - Output layer: 5 units (for 5 classes) with softmax activation	3,454,597 parameters, with 3,213,341 coming from the fully connected layers and 240,256 from the convolutional layers.

B. Training and Testing Time

As indicated in Table 4.2, the training time for the CNN model is significantly higher than that of the ORB model, with a notable difference of 5378.81s. This is primarily due to the complex nature of CNN, which involve intensive computations across multiple deep layers for each epoch. The training process requires not only convolution operations but also backpropagation to update weights, contributing to longer training times. Conversely, the ORB model, with its simpler architecture, requires fewer computations during training. Meanwhile, the testing time for both models is approximately the same.

C. Average CPU and Memory Usage

To measure the average CPU and memory usage, a continuous monitoring approach is employed during both the training and testing phases. The monitor_resources function utilizes the psutil library from Python to track the CPU and memory usage of the current process. Memory usage is measured in megabytes (MB) using the memory_info().rss attribute of the psutil.Process object. During both phases, CPU and memory usage are logged in real-time into shared lists, cpu_usages and memory_usages, respectively. The average values for these metrics are then calculated by taking the mean of the collected data.

As shown in Table 4.2, the average CPU usage for the CNN and ORB models is similar, with CNN utilizing 68.27% and ORB utilizing 71.26%. This slight difference suggests that the ORB model has comparable computational demands to the CNN model during execution, which is attributed to the multi-threading process used in ORB for feature extraction. Regarding memory usage, the ORB model consumes 7673.09 MB, which exceeds CNN's 4801.76 MB. This increased memory consumption is also due to the multi-threading process in ORB. However, when considering the training and testing times for each model, the CNN model occupies CPU and memory for approximately 1.6 hours, while the ORB model only occupies resources for 464.55 seconds (385.53s for training and 79.02s for testing). This highlights that, despite the greater memory usage of the ORB model compared to the CNN model, it utilizes resources more efficiently and within much shorter timeframes.

Table 4.2: Comparison of computational overhead between CNN and ORB models

Aspects	**CNN**	**ORB**
Total trainable parameters	3,454,597	1,028,549
Training time (in seconds)	5764.34	385.53
Testing time (in seconds)	79.43	79.02
Average CPU usage (in percentage)	68.27	71.26
Average Memory usage (in MB)	4801.76	7673.09

The experimental findings reveal that, although the ORB model exhibits a 1.99% lower accuracy compared to the CNN model, its significantly reduced computational expense makes it the optimal choice for traffic classification tasks. This efficiency is evident in the model's streamlined architecture. Unlike the CNN model, which demands extensive computational resources due to its deeper architecture, the ORB model relies on straightforward feature-based extraction followed by classification with a simple neural network. Moreover, the ORB model demonstrates a significant reduction in training time, requiring only a fraction of the time taken by the CNN model, 385.53s compared to 5764.34s. While its memory usage is greater than that of the CNN model due to multi-threading, it remains more efficient overall in handling large-scale traffic classification scenarios where computational efficiency directly impacts real-time performance. Therefore, despite the difference in accuracy, the ORB model is well-suited for tasks demanding both speed and reduced resource utilization.

4.3.3 Performance Evaluation of ORB model on the CIC-DDoS2019 Dataset

The ORB model demonstrates outstanding performance on our experimental CIC-DDoS2019 dataset, achieving an accuracy of 99.86%. This high accuracy reflects the model's ability to effectively distinguish between various attack types and benign traffic with minimal misclassification.

The confusion matrix in Figure 4.15 highlights the model's strong capability in accurately classifying various traffic types, reflecting its robustness in differentiating between benign and malicious flows. For attack classes such as SYN, UDP, MSSQL, and LDAP, the model demonstrates perfect classification accuracy. Specifically, SYN traffic shows only two misclassifications out of 2,047 instances, while LDAP achieves similarly exceptional results with just one misclassification out of 2,000 instances. These low misclassification rates indicate the model's effectiveness in accurately identifying the majority of attack traffic with minimal errors. The classification of benign traffic presents a slight challenge, with 9 misclassifications out of 289 instances; however, the model achieves zero false positives, further underscoring its reliability.

The classification report in Figure 4.16 provides additional insight into the model's consistent performance across all traffic classes. It achieves perfect precision, recall, and F1-scores for all attack traffic classes. For the benign traffic class, the recall reaches 0.97, which, while slightly lower than that of the attack classes, remains impressive. This is likely due to the limited representation of benign instances in the dataset.

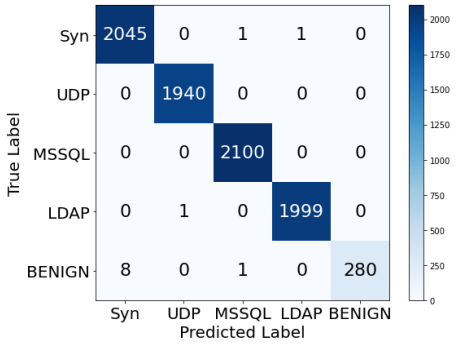


Figure 4.15: Confusion matrix within our experimental CIC-DDoS2019 dataset

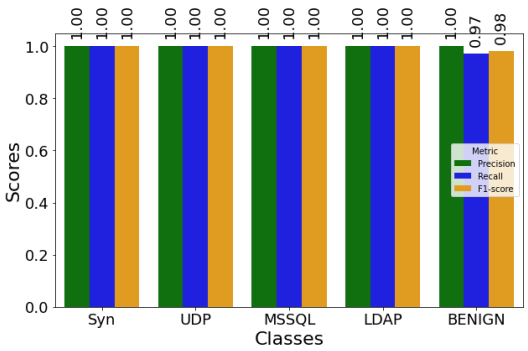


Figure 4.16: Classification reports within our experimental CIC-DDoS2019 dataset

4.4 ORB Model Scalability

The scalability of the ORB model in real-world SDN environments depends on its ability to efficiently handle large traffic volumes across all processing stages, including flow-to-bar chart conversion, feature extraction, and classification. As networks expand and traffic patterns become more complex, real-time processing is essential to avoid latency. Flow-to-bar chart conversion can be particularly challenging in resource-constrained environments, as it, along with feature extraction and classification, can become computationally intensive, especially during peak traffic periods, leading to delays in SDN performance. To address these challenges, several strategies can improve scalability:

- **Optimized flow-to-bar chart conversion:** Implementing efficient algorithms, reducing processing time, and managing memory effectively to prevent slowdowns.
- **Hardware acceleration:** Utilizing GPUs or TPUs to improve both flow-to-bar chart conversion and feature extraction.
- **Traffic sampling techniques:** Applying down-sampling or selecting representative flows to reduce data volume without compromising classification accuracy.
- **Hierarchical data processing:** Prioritizing critical flows for detailed analysis while processing less significant flows in aggregate to balance efficiency and performance.
- **Neural network optimizations:** Applying pruning (reducing unnecessary parameters) and quantization (using lower precision formats) to achieve faster inference and lower memory consumption.

4.4.1 Comparison

Table 4.3 provides an overview of various models employed for traffic classification and anomaly detection. Many studies, particularly those utilizing CNNs, focus primarily on identifying DDoS attacks. For instance, the CNN model in [156] achieves an accuracy of 98.98% using packet data in SDN environments. Similarly, another CNN-based approach applied to simulated mobile cloud computing reports a notable 99% accuracy for DDoS detection [157]. However, this model requires further refinement in architecture and data preprocessing methods, and its narrow focus limits its capability to detect attacks other than DDoS. The SAE-MLP model [158], tested on a custom SDN dataset, achieves an impressive accuracy of 99.75% for DDoS detection, but its limited scope also reduces its effectiveness in identifying other attack types. In contrast, the proposed ORB model not only achieves a superior detection accuracy of 99.92% for DDoS attacks on the InSDN dataset but is also capable of identifying a broader range of attacks beyond DDoS.

For intrusion traffic detection, the methods presented in Table 4.3 exhibit certain limitations that hinder their overall effectiveness. Sparse autoencoders [159] achieved an average accuracy of 98.5% on the NSL-KDD and CICIDS2017 datasets, which are not specific to SDN environments, reducing their relevance for SDN-based networks. Additionally, their multi-phase approach, while enhancing

accuracy, introduces significant overhead due to its complexity. Similarly, CNN, DNN, RNN, LSTM, and GRU models [143], with accuracies ranging from 97.78% to 98.63% on the NSL-KDD dataset, face the same limitation, as they are not designed for SDN, which restricts their performance in SDN environments. The Vision Transformer model [103], with 98.5% accuracy on the CICIDS2017 dataset and 96.3% on the UNSW-NB15 dataset, also struggles with the need for extensive preprocessing, which reduces its applicability in real-time scenarios. Furthermore, a CNN model [107] using the NSL-KDD dataset achieves a low accuracy of 88.82%. This model is also constrained by preprocessing overhead and its reliance on a general dataset, limiting its effectiveness for intrusion detection in specialized environments like SDN. In comparison, our proposed model surpasses these models' accuracies with 99.86% on the CIC-DDoS2019 dataset while maintaining lower computational expense.

To conclude this comparison, in 2024, the authors in [160] developed a traffic classification model for SDN using BiLSTM, LSTM, BiGRU, and GRU on two datasets, InSDN and VPN-nonVPN [161], to improve QoS and security. GRU achieved an accuracy of 99.65%, but as mentioned in [162], the model faced challenges related to training complexity and computational costs. On the other hand, our ORB model achieves 97.14% accuracy on the InSDN dataset while offering greater computational efficiency. It also demonstrates exceptional performance on the CIC-DDoS2019 dataset, with an accuracy of 99.86%. With its superior performance and reduced preprocessing requirements, the ORB model is better suited than existing models for real-time applications with low resource overhead.

Table 4.3: Performance comparison of traffic classification methods: ORB model vs previous studies

Ref.	Model	Traffic Class	Dataset(s)	Acc.	Comment
[156]	CNN	DDoS	Packet Data in SDN	98.98%	High accuracy, limited to DDoS detection.
[159]	SAE	Intrusions	NSL-KDD, CICIDS2017	98.5%	Complexity increases with multi-phase processing. Datasets not specific to SDN intrusions.
[143]	CNN, DNN, RNN, LSTM, GRU	Intrusions	NSL-KDD	CNN: 98.63%, DNN: 98.53%, RNN: 98.13%, LSTM: 98.04%, GRU: 97.78%	Resource-intensive; dataset not tailored for SDN-based intrusion analysis.
[103]	ViT	Intrusions	CICIDS2017, UNSW-NB15	98.5% (CICIDS2017), 96.3% (UNSW-NB15)	Extensive preprocessing limits real-time applicability.
[107]	CNN	Intrusions	NSL-KDD	88.82%	Novel preprocessing but computationally expensive.
[157]	CNN	DDoS	Simulated Mobile Cloud Computing	99%	Computationally intensive, requiring model enhancements.
[158]	SAE-MLP	DDoS	Custom Dataset for SDN	99.75%	Narrow focus on DDoS detection.
[160]	CNN, DNN, RNN, LSTM, GRU	Multimedia, VoIP, instant messaging, attacks	InSDN, ISCX-VPN-NoVPN	GRU: 99.65%	Issues with training complexity and high computational requirements.
Current Work	ORB with NN	Probe, DoS, DDoS, Normal, BFA	InSDN, CIC-DDoS2019	InSDN: 97.14%, DDoS Detection: 99.92%; CIC-DDoS2019: 99.86%.	Reduces computational overhead while maintaining high accuracy.

4.4.2 Novelty of the ORB Model

Building on insights from prior studies and the experimental results of the ORB model, Table 4.4 emphasizes the unique aspects of our approach compared to existing methods. While earlier research has advanced traffic classification through various techniques, our study introduces a feature-based method that achieves a balance between computational efficiency and performance. Specifically, it combines ORB with a neural network, providing a more resource-efficient alternative to computationally intensive models such as CNN, ViT, LSTM, and GRU.

Furthermore, we present an innovative flow-to-image conversion technique that transforms traffic data into bar chart images, while other studies typically rely on packet-level or byte-level image conversion. This approach, compared against IGTD using Euclidean and Manhattan distances, demonstrates promising results. Unlike methods focused solely on accuracy, our approach balances performance with resource efficiency, making it highly suitable for SDN environments with constrained computational resources.

Table 4.4: Novel contributions in traffic classification: ORB Model vs. previous studies

Aspect	**This Study**	**Previous Studies**
Machine Learning Technique	Combines ORB with neural network for efficient traffic classification.	Unlike our approach, existing models such as CNN, ViT, LSTM, and GRU use alternative mechanisms for attack detection and flow analysis (e.g., [103, 143, 156, 157]).
Feature Extraction	Utilizes ORB for efficient feature extraction from flow data.	Generally relies on deep learning for feature extraction (e.g., [107, 157]).
Flow-to-Image Conversion	Introduces a new method for converting traffic data into images, transforming each flow into a bar chart image.	Other studies use different methods; for example, [107] converts packet data into images, while [103] transforms flow data into RGB images, and [156] processes network packets and generates traffic images by representing each byte as a pixel.
Computational Efficiency	The approach optimizes both accuracy and resource usage, enabling real-time attack detection without overloading SDN resources, thereby enhancing security.	Focuses on overall accuracy with less emphasis on computational constraints in SDN environments (e.g., [103, 143, 156]).

4.5 Conclusion

This chapter introduces a novel image-based method for traffic classification to enhance SDN security. Our approach provides an alternative to the computationally intensive CNN by leveraging ORB or SIFT for feature extraction, combined with a neural network for traffic classification in SDN environments. Experimental results on the InSDN dataset indicate that CNN achieves higher accuracy and robustness across various data transformation methods. However, the ORB-based model, utilizing flow-to-bar chart conversion, delivers competitive performance while significantly reducing computational overhead. Furthermore, the proposed ORB-based model achieves outstanding results on the CIC-DDoS2019 dataset, attaining an accuracy of 99.86%. These findings highlight its potential as an efficient solution for traffic classification in SDN environments. The next chapter conducts a comparative analysis of our three proposed approaches based on different key aspects.

Chapter 5

Synthesis of Our Traffic Classification and Intrusion Detection Approaches in SDN

This chapter analyzes the strengths and limitations of our three proposed approaches for traffic classification and intrusion detection in SDN environments. Each method uses distinct techniques for processing network traffic, varying in design complexity, detection accuracy, and computational resource requirements, making them suitable for different deployment contexts.

The comparison focuses on key dimensions: the effectiveness of feature engineering strategies, computational efficiency, scalability in large or dynamic SDN environments, the types of attacks targeted, and the performance of the machine learning techniques. By evaluating these aspects, we aim to highlight the trade-offs and advantages of each approach, offering insights into their applicability for real-world SDN deployments and guiding future improvements.

5.1 Feature Engineering

This section examines the feature engineering techniques used in our three contributions, each employing different strategies for processing network traffic data in SDN environments, including clustering-based, embedding-based, and image-based approaches.

The first contribution involves a hybrid model combining multiple K-Means models with a Gaussian Naïve Bayes classifier. For each value of k, representing the number of clusters, separate K-Means models are trained independently. One K-Means model is trained for each selected feature, such as flow duration or packets per second, capturing individual feature patterns without accounting for inter-feature dependencies. The trained K-Means models are used to assign clusters, which are then used as input features to train a Gaussian Naïve Bayes classifier.

The second contribution investigates Word2Vec embeddings to enhance classification accuracy and efficiency. Multiple K-Means models are trained independently, with each model clustering a selected numerical feature. The resulting clusters are then converted into categorical labels in the format ”At-

tribute_ClusterX", where X represents the assigned cluster and "Attribute" denotes the feature name. These categorical values are combined into lists of strings, forming a text-like representation for each flow. A Word2Vec model is then trained on these lists to capture semantic relationships between the categorical features. To obtain a structured numerical representation of the flow, the mean of its word embeddings is computed, capturing overall characteristics while preserving contextual relationships between features.

The third contribution focuses on feature extraction from images generated using flow data. Three methods are employed for image generation: (1) transforming flow features into bar charts, (2) applying IGTD with Euclidean distance, and (3) applying IGTD with Manhattan distance. After data preprocessing, feature extraction is performed using either ORB, SIFT, or a CNN. ORB extracts 500 keypoints per image, each described by a 32-dimensional descriptor, resulting in a flattened 16,000-dimensional feature vector. In contrast, SIFT produces 128-dimensional descriptors for each keypoint, leading to 64,000-dimensional feature vectors. Alternatively, CNN automatically learns feature representations through its convolutional layers.

5.2 Computational Overhead

The computational overhead of the methods discussed varies significantly across the three contributions, reflecting differences in complexity and resource requirements.

Contribution 1 presents a computationally efficient approach through parallel clustering in a one-dimensional space and the use of Naïve Bayes, both of which require minimal resources. The training of K-Means models is performed independently and in parallel, enabling faster processing. However, as the number of clusters increases, the training time also rises due to the additional iterations required for centroid updates and data point assignments. For instance, on the InSDN dataset, training time increases from 3.28 seconds for 5 clusters to 29.59 seconds for 40 clusters. Similarly, on the CIC-DDoS2017 dataset, training time rises from 5.20 seconds for 3 clusters to 124.06 seconds for 93 clusters. Despite this increase, the overall computational requirements remain significantly lower than those of more complex models, making the approach suitable for real-time applications.

Contribution 2 introduces a method with higher computational requirements, primarily due to the integration of Word2Vec and a neural network classifier. Word2Vec, a neural network-based model applied to encode categorical features as vector representations, adds computational complexity, particularly for large datasets. Neural networks further increase training time and memory usage compared to traditional methods like Naïve Bayes due to their multiple layers and weight optimizations. The combination of multiple K-Means models, Word2Vec, and a neural network classifier substantially increases computational overhead. For the InSDN dataset, training times without feature selection range from 328.79 to 841.08 seconds, whereas feature selection reduces this range to 158.40–449.53 seconds, demonstrating its effectiveness in reducing computational costs.

Contribution 3 introduces a computationally efficient image-based model that combines ORB for keypoint detection and descriptor computation with a neural network for classification. Compared to a CNN model, the ORB model requires 1,028,549 trainable parameters, whereas CNN's deeper architecture results in 3,454,597 parameters, leading to a 5,378.81-second longer training time (5,764.34s vs. 385.53s). Despite the ORB model's multi-threading leading to increased memory usage, it remains significantly less computationally demanding than the CNN model, making it more suitable for real-time applications.

5.3 Scalability

Contribution 1 is highly scalable due to its lightweight design, which minimizes resource usage. While the number of clusters may impact processing time, the model remains effective for real-time flooding DDoS detection in dynamic SDN environments. Scalability can be further improved using Mini-Batch K-Means to reduce training time and memory consumption. Additionally, GPU acceleration can enhance performance, while distributed computing frameworks like Apache Spark enable scaling across multiple machines.

The scalability of the second contribution is limited by the computational complexity of the integrated models, especially when processing large datasets or real-time traffic. However, this limitation can be addressed through various optimizations. Pruning and quantization, for instance, reduce neural network size and processing time, while feature selection improves both computational and memory efficiency. Furthermore, techniques from contribution 1 can be leveraged to further optimize this approach, enhancing its suitability for real-time applications. Together, these optimizations improve scalability while reducing computational overhead.

The ORB model in contribution 3 balances accuracy and computational efficiency, making it better suited than the CNN model for resource-constrained environments. However, its scalability is limited by the complexity of the model, particularly when converting flow data to bar charts during high traffic volumes in real-time SDN environments. To enhance scalability, several strategies can be employed, such as hardware acceleration (GPUs/TPUs), traffic sampling, and hierarchical data processing.

Table 5.1 provides a comparative performance analysis, highlighting the trade-offs between accuracy, scalability, and computational cost.

Table 5.1: Comparative analysis of proposed models for cyberattack detection in SDN environments

Model	Detected Attacks	Datasets	Feature Engineering	Computational Overhead	Scalability	Model Performance	Drawbacks
Multiple K-Means + Naïve Bayes	Flooding DDoS	InSDN, CIC-DDoS2017	Clusters assigned by multiple K-Means models.	Low computational overhead, though training time increases with the number of clusters. However, overall resource demands remain manageable compared to more complex methods.	Scales efficiently with low resource utilization. Mini-Batch K-Means, GPU acceleration, and distributed computing can improve large-scale performance.	InSDN: at k=32 clusters, 99.98% accuracy, precision: 99.95%, recall: 99.99%, F1-score: 99.97%. CIC-DDoS2017: at k=86 clusters, 99.70% accuracy, precision: 99.25%, recall: 99.57%, F1-score: 99.41%.	Requires testing multiple k values to determine the optimal model; Naïve Bayes assumes feature independence, which may not always hold in network traffic.
Multiple K-Means + Word2Vec + Neural Network	DoS, DDoS, BFA, and Probe	InSDN, CIC-DDoS2019	Multiple K-Means models are used to cluster numerical features, transforming them into categorical representations. Each flow is then represented as a sequence of categorical values, which are embedded using Word2Vec to capture semantic relationships. The mean of these embeddings produces a compact flow representation.	High computational overhead arises from the use of multiple K-Means models, Word2Vec, and neural networks, with training times ranging from 328.79 to 841.08 seconds. Applying feature selection reduces this time by approximately 50%.	Scalability is initially constrained; however, optimizations such as GPU acceleration, pruning, quantization, and feature selection can significantly improve large-scale data handling and real-time performance.	InSDN: at k=115 (no feature selection), 99.97% accuracy, F1-score: 0.99–1.00; with feature selection, at k=60, 98.92% accuracy, F1-score: 0.97–1.00. CIC-DDoS2019: at k=280, 98.65% accuracy, F1-score: 0.98–1.00 (except UDPLag: 0.92).	Requires testing multiple k values to identify the best-performing model; High Computational Overhead.
ORB + NN	DoS, DDoS, BFA, and Probe	InSDN, CIC-DDoS2019	ORB extracts up to 500 keypoints per image and computes their descriptors.	Reduced computational cost compared to CNN, leveraging the efficiency of the fast ORB algorithm and a simple neural network architecture.	Scalability is constrained by data preprocessing overhead but can be significantly improved through optimized flow-to-bar chart processing and the use of hardware accelerators, such as GPUs or TPUs.	InSDN: 97.14% accuracy, F1: 0.96–1.00 (except BFA: 0.87). CIC-DDoS2019: 99.86% accuracy with minimal misclassifications.	Flow-to-bar chart conversion in the data preprocessing phase can be time-consuming and introduces an additional computational burden.

5.4 Conclusion

This chapter compares three SDN-based traffic classification approaches, each utilizing distinct feature engineering and machine learning techniques. Contribution 1 offers a scalable and lightweight solution for Flooding DDoS detection, making it highly suitable for real-time environments. Contribution 2 extends the classification scope to a broader range of attacks, including DoS, DDoS, BFA, and Probe, but incurs significant computational overhead, requiring optimizations to improve scalability for large-scale applications. Contribution 3, utilizing an image-based approach with ORB feature extraction, strikes a better balance between accuracy and efficiency compared to CNN, though data preprocessing overhead presents scalability challenges. The choice of approach depends on application-specific requirements, including target attack types, resource availability, and real-time processing constraints. Optimizations such as pruning, quantization, and hardware acceleration can further enhance the performance of these models in large-scale and real-time deployments.

General Conclusion and Future Directions

This work addresses the challenges of SDN security by introducing innovative feature-based methods to improve traffic classification and intrusion detection, with careful consideration of computational overhead. The first contribution combines K-Means clustering with Naïve Bayes classification, focusing on simplicity and computational efficiency. It is well-suited for real-time applications, particularly for detecting Flooding DDoS attacks, offering high accuracy with low resource consumption.

The next contribution builds on this approach by further exploring the effectiveness of one-dimensional K-Means clustering. Multiple K-Means models cluster numerical features, converting them into categorical representations. Each flow is represented as a sequence of these categorical values, which are then embedded using Word2Vec to capture their semantic relationships. The average of these embeddings creates a compact representation of the flow, which is subsequently classified using a neural network. This method enhances the ability to capture complex patterns across multiple attack types, including DoS, DDoS, BFA, and Probe. While it incurs high computational overhead, optimizations such as feature selection, neural network pruning, and quantization can significantly reduce this cost, making the solution more adaptable for large-scale SDN deployments.

The third contribution also targets a broader range of attacks and introduces a novel flow-to-image conversion method, where network flows are transformed into bar chart images. Features are extracted from these images using ORB keypoint detection and descriptor computation, followed by classification with a lightweight neural network. Compared to CNN-based approaches, this method achieves a better trade-off between accuracy and computational efficiency. Further optimizations, such as down-sampling and hardware acceleration using GPUs or TPUs, can enhance scalability and real-time performance.

By integrating techniques from natural language processing and computer vision, this work strengthens the feature extraction and classification processes, offering new perspectives for efficient SDN traffic analysis and providing promising solutions for real-time security in SDN environments.

Our contributions open several research directions, including enhancements to existing methods and the exploration of alternative approaches. Future research could focus on:

- **Combining the first two methods**: The main benefit of this hybrid approach is that it helps reduce false negatives. Since both methods employ multiple K-Means models, a promising direction is to

combine them into a unified approach. One effective strategy is to implement a confidence-based hybrid classification system. First, clustering is applied using K-Means models. Then, Naïve Bayes serves as the initial classifier. If its confidence score is high (e.g., $\geq 70\%$), the classification is accepted immediately. Otherwise, the flow is processed by Word2Vec and the neural network for further evaluation. The confidence score is determined based on the probability output of the Naïve Bayes classifier.

- **Testing alternative clustering algorithms for the first two methods**: Since identifying the best-performing model for the two K-Means-based approaches requires testing multiple cluster numbers, an alternative is to use algorithms that do not require a predefined number of clusters, such as DBSCAN or Mean Shift.

- **Optimizing the ORB model**: Instead of transforming network flows into bar chart images, the flow can be converted into a pixel matrix that mirrors the structure of the bar chart. ORB (or an adapted version of ORB) can then be used to extract features from these matrices. Additionally, implementing batch processing for flows can further improve computational efficiency. For example, 16 flows can be converted into 16 pixel matrices simultaneously, from which features are extracted. These features are then used to train the neural network. The training process is repeated iteratively for each batch of 16 flows, enabling efficient handling of flow data and potentially enhancing the processing efficiency of the ORB model.

Bibliography

[1] H. Yzzogh and H. Benaboud, "A comprehensive overview of machine learning for intrusion detection in software-defined networking," *Innovations Syst Softw Eng*, 2025, 10.1007/s11334- 025-00604-6.

[2] "Flooding distributed denial of service detection in software-defined networking using k-means and naıve bayes," *IJECE*, vol. 15 (1), pp. 817–826, February, 2025, doi: 10.11591/ijece.v15i1.pp817-826.

[3] "Enhancing sdn security with a feature-based approach using multiple k-means, word2vec, and neural network," *Bulletin EEI*, vol. 14 (2), p. 1456–1467, April, 2025, doi: 10.11591/eei.v14i2.8834.

[4] "Evaluating orb and sift with neural network as alternatives to cnn for traffic classification in sdn environments,," *IEEE Access*, vol. 13, p. 51484–51498, 2025, doi: 10.1109/ACCESS.2025.3553449.

[5] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, "Software-defined networking (sdn): Layers and architecture terminology," 2015, rFC7426, January 2015, available at: https://datatracker.ietf.org/doc/html/rfc7426.

[6] T. Mizrahi, N. Sprecher, E. Bellagamba, and Y. Weingarten, "An overview of operations, administration, and maintenance (oam) tools," 2014, rFC 7276, June 2014, available at: http://www.rfc-editor.org/info/rfc7276.

[7] "Openflow switch specification," 2015, available at: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf. Accessed: 23-Feb-2024.

[8] C. Qi, J. Wu, H. Hu, G. Cheng, W. Liu, J. Ai, and C. Yang, "An intensive security architecture with multi-controller for sdn," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2016, pp. 401–402, doi: 10.1109/INFCOMW.2016.7562109.

[9] C. Röpke and T. Holz, "Preventing malicious sdn applications from hiding adverse network manipulations," in *Proceedings of the 2018 Workshop on Security in Softwarized Networks: Prospects and Challenges*, 2018, pp. 40–45, doi: 10.1145/3229616.3229620.

[10] C. Lee, C. Yoon, S. Shin, and S. K. Cha, "Indago: A new framework for detecting malicious sdn applications," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, 2018, pp. 220–230, doi: 10.1109/ICNP.2018.00031.

[11] S. Scott-Hayward, C. Kane, and S. Sezer, "Operationcheckpoint: Sdn application control," in *2014 IEEE 22nd International Conference on Network Protocols*, 2014, pp. 618–623, doi: 10.1109/ICNP.2014.98.

[12] P. A. Porras, S. Cheung, M. W. Fong, K. Skinner, and V. Yegneswaran, "Securing the software defined network control layer." in *Ndss*, 2015, doi: 10.14722/ndss.2015.23222.

[13] X. Wen, Y. Chen, C. Hu, C. Shi, and Y. Wang, "Towards a secure controller platform for openflow applications," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 171–172, doi: 10.1145/2491185.2491212.

[14] Y. Tseng, Z. Zhang, and F. Naït-Abdesselam, "Controllersepa: A security-enhancing sdn controller plug-in for openflow applications," in *2016 17th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, 2016, pp. 268–273, doi: 10.1109/PDCAT.2016.064.

[15] H. Li, P. Li, S. Guo, and A. Nayak, "Byzantine-resilient secure software-defined networks with multiple controllers in cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 4, pp. 436–447, 2014, doi: 10.1109/TCC.2014.2355227.

[16] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning network visibility in software-defined networks: New attacks and countermeasures." in *Ndss*, vol. 15, 2015, pp. 8–11, doi: 10.14722/ndss.2015.23283.

[17] M. Ambrosin, M. Conti, F. De Gaspari, and R. Poovendran, "Lineswitch: Tackling control plane saturation attacks in software-defined networking," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1206–1219, 2017, doi: 10.1109/TNET.2016.2626287.

[18] B. Han, X. Yang, Z. Sun, J. Huang, and J. Su, "Overwatch: a cross-plane ddos attack defense framework with collaborative intelligence in sdn," *Security and Communication Networks*, vol. 2018, no. 1, p. 9649643, 2018, doi: 10.1155/2018/9649643.

[19] S. Deng, X. Gao, Z. Lu, and X. Gao, "Packet injection attack and its defense in software-defined networks," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 3, pp. 695–705, 2018, doi: 10.1109/TIFS.2017.2765506.

[20] A. S. Alshra'a and J. Seitz, "Using inspector device to stop packet injection attack in sdn," *IEEE Communications Letters*, vol. 23, no. 7, pp. 1174–1177, 2019, doi: 10.1109/LCOMM.2019.2896928.

[21] H. Jo, J. Nam, and S. Shin, "Nosarmor: Building a secure network operating system," *Security and Communication Networks*, vol. 2018, no. 1, p. 9178425, 2018, doi: 10.1155/2018/9178425.

[22] J. Noh, S. Lee, J. Park, S. Shin, and B. B. Kang, "Vulnerabilities of network os and mitigation with state-based permission system," *Security and Communication Networks*, vol. 9, no. 13, pp. 1971–1982, 2016, doi: 10.1002/sec.1369.

[23] M. Brooks and B. Yang, "A man-in-the-middle attack against opendaylight sdn controller," in *Proceedings of the 4th annual ACM conference on research in information technology*, 2015, pp. 45–49, doi: 10.1145/2808062.2808073.

[24] K. Zhang and X. Qiu, "Cmd: A convincing mechanism for mitm detection in sdn," in *2018 IEEE International Conference on Consumer Electronics (ICCE)*, 2018, pp. 1–6, doi: 10.1109/ICCE.2018.8326334.

[25] A. Aseeri, N. Netjinda, and R. Hewett, "Alleviating eavesdropping attacks in software-defined networking data plane," in *Proceedings of the 12th Annual Conference on Cyber and Information Security Research*, 2017, pp. 1–8, doi: 10.1145/3064814.3064832.

[26] G. Shang, P. Zhe, X. Bin, H. Aiqun, and R. Kui, "Flooddefender: Protecting data and control plane resources under sdn-aimed dos attacks," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9, doi: 10.1109/INFOCOM.2017.8057009.

[27] M. Zhang, J. Bi, J. Bai, and G. Li, "Floodshield: Securing the sdn infrastructure against denial-of-service attacks," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2018, pp. 687–698, doi: 10.1109/TrustCom/BigDataSE.2018.00101.

[28] D. M. Ferrazani Mattos and O. C. M. B. Duarte, "Authflow: authentication and access control mechanism for software defined networking," *annals of telecommunications*, vol. 71, pp. 607–615, 2016, doi: 10.1007/s12243-016-0505-z.

[29] M. Myint Oo, S. Kamolphiwong, T. Kamolphiwong, and S. Vasupongayya, "Advanced support vector machine-(asvm-) based detection for distributed denial of service (ddos) attack on software defined networking (sdn)," *Journal of Computer Networks and Communications*, vol. 2019, no. 1, p. 8012568, 2019, doi: 10.1155/2019/8012568.

[30] S. Wang, S. Chandrasekharan, K. Gomez, S. Kandeepan, A. Al-Hourani, M. R. Asghar, G. Russello, and P. Zanna, "Secod: Sdn secure control and data plane algorithm for detecting and defending against dos attacks," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–5, doi: 10.1109/NOMS.2018.8406196.

[31] A. Azzouni, O. Braham, T. M. Trang Nguyen, G. Pujolle, and R. Boutaba, "Fingerprinting openflow controllers: The first step to attack an sdn control plane," in *2016 IEEE Global Communications Conference (GLOBECOM)*, 2016, pp. 1–6, doi: 10.1109/GLOCOM.2016.7841843.

[32] M. Zhang, J. Hou, Z. Zhang, W. Shi, B. Qin, and B. Liang, "Fine-grained fingerprinting threats to software-defined networks," in *2017 IEEE Trustcom/BigDataSE/ICESS*, 2017, pp. 128–135, doi: 10.1109/Trustcom/BigDataSE/ICESS.2017.229.

[33] Y. Qian, W. You, and K. Qian, "Openflow flow table overflow attacks and countermeasures," in *2016 European Conference on Networks and Communications (EuCNC)*, 2016, pp. 205–209, doi: 10.1109/EuCNC.2016.7561033.

[34] T. Xu, D. Gao, P. Dong, C. H. Foh, and H. Zhang, "Mitigating the table-overflow attack in software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 1086–1097, 2017, doi: 10.1109/TNSM.2017.2758796.

[35] E. Fenil and P. Mohan Kumar, "Shchain_3d-resnet: Sharding blockchain with 3d-residual network (3d-resnet) deep learning model for classifying ddos attack in software defined network," *Symmetry*, vol. 14, no. 6, p. 1254, 2022, doi: 10.3390/sym14061254.

[36] R. K. Chouhan, M. Atulkar, and N. K. Nagwani, "A framework to detect ddos attack in ryu controller based software defined networks using feature extraction and classification," *Applied Intelligence*, vol. 53, no. 4, pp. 4268–4288, 2023, doi: 10.1007/s10489-022-03565-6.

[37] J. Wang and L. Wang, "Sdn-defend: a lightweight online attack detection and mitigation system for ddos attacks in sdn," *Sensors*, vol. 22, no. 21, p. 8287, 2022, doi: 10.3390/s22218287.

[38] S. D. Kebede, B. Tiwari, V. Tiwari, and K. Chandravanshi, "Predictive machine learning-based integrated approach for ddos detection and prevention," *Multimedia Tools and Applications*, vol. 81, no. 3, pp. 4185–4211, 2022, doi: 10.1007/s11042-021-11740-z.

[39] A. H. Janabi, T. Kanakis, and M. Johnson, "Convolutional neural network based algorithm for early warning proactive system security in software defined networks," *IEEE Access*, vol. 10, pp. 14 301–14 310, 2022, doi: 10.1109/ACCESS.2022.3148134.

[40] V. Deepa, K. M. Sudar, and P. Deepalakshmi, "Design of ensemble learning methods for ddos detection in sdn environment," in *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)*. IEEE, 2019, pp. 1–6, doi: 10.1109/ViTECoN.2019.8899682.

[41] D. Firdaus, R. Munadi, and Y. Purwanto, "Ddos attack detection in software defined network using ensemble k-means++ and random forest," in *2020 3rd International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, 2020, pp. 164–169, doi: 10.1109/ISRITI51436.2020.9315521.

[42] T. Zoppi, A. Ceccarelli, and A. Bondavalli, "Unsupervised algorithms to detect zero-day attacks: Strategy and application," *Ieee Access*, vol. 9, pp. 90 603–90 615, 2021, doi: 10.1109/ACCESS.2021.3090957.

[43] "DARPA1998 Dataset," https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset, 1998, [Accessed: 20-Feb-2024].

[44] "KDD99 Dataset," http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, 1999, [Accessed: 26-Feb-2024].

[45] "The caida ucsd anonymized internet traces," 2016, accessed: 10-Feb-2024. [Online]. Available: https://www.caida.org/data/passive/passive_2016_dataset.xml

[46] "The caida ucsd ddos attack 2007 dataset," Accessed: 10-Feb-2024, 2017, [Online]. Available: http://www.caida.org/data/passive/ddos-20070804_dataset.xml.

[47] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "Nsl-kdd dataset," Available online, accessed: 10 November 2023. [Online]. Available: https://www.kaggle.com/datasets/hassan06/nslkdd

[48] "CTU Malware Capture Botnets in CTU-13 dataset," 2011, accessed: 21-Sep-2023. [Online]. Available: https://mcfp.felk.cvut.cz/publicDatasets/

[49] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, 2015, pp. 1–6, doi: 10.1109/MilCIS.2015.7348942.

[50] C. Orsini, A. King, D. Giordano, V. Giotsas, and A. Dainotti, "Bgpstream: a software framework for live and historical bgp data analysis," in *Proceedings of the 2016 Internet Measurement Conference*, 2016, pp. 429–444, doi: 10.1145/2987443.2987482.

[51] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Intrusion detection evaluation dataset (cic-ids2017)," 2017, accessed: 19-Dec-2023. [Online]. Available: https://www.unb.ca/cic/datasets/ids-2017.html

[52] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani *et al.*, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," *ICISSp*, vol. 1, pp. 108–116, 2018, doi: 10.5220/0006639801080116.

[53] "CSE-CIC-IDS2018 on AWS - A collaborative project between the Communications Security Establishment (CSE) & the Canadian Institute for Cybersecurity (CIC)," 2018, accessed: 26-Oct-2023. [Online]. Available: https://www.unb.ca/cic/datasets/ids-2018.html

[54] "DDoS Evaluation Dataset (CIC-DDoS2019)," 2019, accessed: 8-Sep-2024. [Online]. Available: https://www.unb.ca/cic/datasets/ddos-2019.html

[55] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (ddos) attack dataset and taxonomy," in *2019 international carnahan conference on security technology (ICCST)*. IEEE, 2019, pp. 1–8, doi: 10.1109/CCST.2019.8888419.

[56] M. S. Elsayed, N.-A. Le-Khac, and A. D. Jurcut, "Insdn: A novel sdn intrusion dataset," *IEEE access*, vol. 8, pp. 165 263–165 284, 2020, doi: 10.1109/ACCESS.2020.3022633.

[57] S. M. Elsayed, N.-A. Le-Khac, and A. Jurcut, "Index of /datasets/sdn," Available online: https://aseados.ucd.ie/datasets/SDN/, accessed: 07 January 2024.

[58] A. Biswas, S. Dutta, T. L. Turton, and J. Ahrens, "Sampling for scientific data analysis and reduction," in *In Situ Visualization for Computational Science*. Springer, 2022, pp. 11–36, doi: 10.1007/978-3-030-81627-8_2.

[59] D. J. White, "Epsilon-dominating solutions in mean-variance portfolio analysis," *European Journal of Operational Research*, vol. 105, no. 3, pp. 457–466, 1998, doi: 10.1016/S0377-2217(97)00056-8.

[60] M. Di Mauro, G. Galatro, G. Fortino, and A. Liotta, "Supervised feature selection techniques in network intrusion detection: A critical review," *Engineering Applications of Artificial Intelligence*, vol. 101, p. 104216, 2021, doi: 10.1016/j.engappai.2021.104216.

[61] M. Dash, K. Choi, P. Scheuermann, and H. Liu, "Feature selection for clustering - a filter solution," in *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, 2002, pp. 115–122, doi: 10.1109/ICDM.2002.1183893.

[62] J. Maldonado, M. C. Riff, and B. Neveu, "A review of recent approaches on wrapper feature selection for intrusion detection," *Expert Systems with Applications*, vol. 198, p. 116822, 2022, doi: 10.1016/j.eswa.2022.116822.

[63] K. A. ElDahshan, A. A. AlHabshy, and L. T. Mohammed, "Filter and embedded feature selection methods to meet big data visualization challenges." *Computers, Materials & Continua*, vol. 75, no. 1, 2023, doi: 10.32604/cmc.2023.032287.

[64] B. M. S. Hasan and A. M. Abdulazeez, "A review of principal component analysis algorithm for dimensionality reduction," *Journal of Soft Computing and Data Mining*, vol. 2, no. 1, pp. 20–30, 2021, doi: 10.30880/jscdm.2021.02.01.003.

[65] N. Zhang, Y. Zhong, and S. Dian, "Rethinking unsupervised texture defect detection using pca," *Optics and Lasers in Engineering*, vol. 163, p. 107470, 2023, doi: 10.1016/j.optlaseng.2022.107470.

[66] S. D. Fabiyi, P. Murray, J. Zabalza, and J. Ren, "Folded lda: Extending the linear discriminant analysis algorithm for feature extraction and data reduction in hyperspectral remote sensing," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 14, pp. 12 312–12 331, 2021, doi: 10.1109/JSTARS.2021.3129818.

[67] M. Balamurali, "T-distributed stochastic neighbor embedding," in *Encyclopedia of mathematical geosciences*. Springer, 2023, pp. 1527–1535, doi: 10.1007/978-3-030-85040-1_446.

[68] J. Malik, R. Muthalagu, and P. M. Pawar, "A systematic review of adversarial machine learning attacks, defensive controls, and technologies," *IEEE Access*, vol. 12, pp. 99 382–99 421, 2024, doi: 10.1109/ACCESS.2024.3423323.

[69] M. M. Irfan, S. Ali, I. Yaqoob, and N. Zafar, "Towards deep learning: A review on adversarial attacks," in *2021 International Conference on Artificial Intelligence (ICAI)*, 2021, pp. 91–96, doi: 10.1109/ICAI52203.2021.9445247.

[70] M. P. Novaes, L. F. Carvalho, J. Lloret, and M. L. Proença Jr, "Adversarial deep learning approach detection and defense against ddos attacks in sdn environments," *Future Generation Computer Systems*, vol. 125, pp. 156–167, 2021, doi: 10.1016/j.future.2021.06.047.

[71] N. M. Yungaicela-Naula, C. Vargas-Rosales, and J. A. Perez-Diaz, "Sdn-based architecture for transport and application layer ddos attack detection by using machine and deep learning," *IEEE Access*, vol. 9, pp. 108 495–108 512, 2021.

[72] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015, doi: 10.1109/JPROC.2014.2371999.

[73] Imran, Z. Ghaffar, A. Alshahrani, M. Fayaz, A. M. Alghamdi, and J. Gwak, "A topical review on machine learning, software defined networking, internet of things applications: Research limitations and challenges," *Electronics*, vol. 10, no. 8, p. 880, 2021.

[74] A. Yazdinejadna, R. M. Parizi, A. Dehghantanha, and M. S. Khan, "A kangaroo-based intrusion detection system on software-defined networks," *Computer Networks*, vol. 184, p. 107688, 2021.

[75] O. Benoudifa, A. Ait Wakrime, and R. Benaini, "Autonomous solution for controller placement problem of software-defined networking using muzero based intelligent agents," *Journal of King Saud University - Computer and Information Sciences*, vol. 35, no. 10, p. 101842, 2023, https://doi.org/10.1016/j.jksuci.2023.101842.

[76] S. Kaur and M. Singh, "Hybrid intrusion detection and signature generation using deep recurrent neural networks," *Neural Computing and Applications*, vol. 32, no. 12, pp. 7859–7877, 2020, doi: 10.1007/s00521-019-04187-9.

[77] S. Wang, J. F. Balarezo, K. G. Chavez, A. Al-Hourani, S. Kandeepan, M. R. Asghar, and G. Russello, "Detecting flooding ddos attacks in software defined networks using supervised learning techniques," *Engineering science and technology, an international journal*, vol. 35, p. 101176, 2022, doi: 10.1016/j.jestch.2022.101176.

[78] L. Wang and Y. Liu, "A ddos attack detection method based on information entropy and deep learning in sdn," in *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, vol. 1. IEEE, 2020, pp. 1084–1088, doi: 10.1109/ITNEC48623.2020.9085007.

[79] R. Sanjeetha, A. Raj, K. Saivenu, M. I. Ahmed, B. Sathvik, and A. Kanavalli, "Detection and mitigation of botnet based ddos attacks using catboost machine learning algorithm in sdn environment," *International Journal of Advanced Technology and Engineering Exploration*, vol. 8, no. 76, p. 445, 2021, doi: 10.19101/IJATEE.2021.874021.

[80] H. Polat, O. Polat, and A. Cetin, "Detecting ddos attacks in software-defined networks through feature selection methods and machine learning models," *Sustainability*, vol. 12, no. 3, p. 1035, 2020, doi: 10.3390/su12031035.

[81] B. Nugraha and R. N. Murthy, "Deep learning-based slow ddos attack detection in sdn-based networks," in *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2020, pp. 51–56, doi: 10.1109/NFV-SDN50289.2020.9289894.

[82] W. Zhijun, X. Qing, W. Jingjie, Y. Meng, and L. Liang, "Low-rate ddos attack detection based on factorization machine in software defined network," *IEEE Access*, vol. 8, pp. 17 404–17 418, 2020, doi: 10.1109/ACCESS.2020.2967478.

[83] M. N. Ali, M. Imran, M. S. u. din, and B.-S. Kim, "Low rate ddos detection using weighted federated learning in sdn control plane in iot network," *Applied Sciences*, vol. 13, no. 3, p. 1431, 2023, doi: 10.3390/app13031431.

[84] J. A. Perez-Diaz, I. A. Valdovinos, K.-K. R. Choo, and D. Zhu, "A flexible sdn-based architecture for identifying and mitigating low-rate ddos attacks using machine learning," *IEEE Access*, vol. 8, pp. 155 859–155 872, 2020, doi: 10.1109/ACCESS.2020.3019330.

[85] O. S. Akanji, O. A. Abisoye, and M. A. Iliyasu, "Mitigating slow hypertext transfer protocol distributed denial of service attacks in software defined networks," *Journal of Information and Communication Technology*, vol. 20, no. 3, pp. 277–304, 2021, doi: 10.32890/jict2021.20.3.1.

[86] Ö. Kasim, "A robust dns flood attack detection with a hybrid deeper learning model," *Computers and Electrical Engineering*, vol. 100, p. 107883, 2022, doi: 10.1016/j.compeleceng.2022.107883.

[87] M. Akbari Kohnehshahri, R. Mohammadi, H. Abdoli, and M. Nassiri, "An efficient method for online detection of drdos attacks on udp-based services in sdn using machine learning algorithms," *Mobile Information Systems*, vol. 2022, no. 1, p. 1169035, 2022, doi: 10.1155/2022/1169035.

[88] N. N. Tuan, P. H. Hung, N. D. Nghia, N. V. Tho, T. V. Phan, and N. H. Thanh, "A ddos attack mitigation scheme in isp networks using machine learning based on sdn," *Electronics*, vol. 9, no. 3, p. 413, 2020, doi: 10.3390/electronics9030413.

[89] N. Meti, D. Narayan, and V. Baligar, "Detection of distributed denial of service attacks using machine learning algorithms in software defined networks," in *2017 international conference on advances in computing, communications and informatics (ICACCI)*. IEEE, 2017, pp. 1366–1371, doi: 10.1109/ICACCI.2017.8126031.

[90] A. Sebbar, K. Zkik, Y. Baddi, M. Boulmalf, and D. Kettani, "Mitm detection and defense mechanism cbna-rf based on machine learning for large-scale sdn context," *Journal of Ambient Intelligence and Humanized Computing*, vol. 11, 12 2020, doi: 10.1007/s12652-020-02099-4.

[91] A. Santos da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, "Atlantic: A framework for anomaly traffic detection, classification, and mitigation in sdn," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, 2016, pp. 27–35, doi: 10.1109/NOMS.2016.7502793.

[92] L. Yang, Y. Song, S. Gao, A. Hu, and B. Xiao, "Griffin: Real-time network intrusion detection system via ensemble of autoencoder in sdn," *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 2269–2281, 2022, doi: 10.1109/TNSM.2022.3175710.

[93] A. Dawoud, S. Shahristani, and C. Raun, "Deep learning and software-defined networks: Towards secure iot architecture. internet of things, 3–4, 82–89," 2018, doi: 10.1016/j.iot.2018.09.003.

[94] "Unsupervised deep learning for software defined networks anomalies detection," *Transactions on Computational Collective Intelligence XXXIII*, pp. 167–178, 2019, doi: 10.1007/978-3-662-59540-4_9.

[95] N. T. Van, H. Bao, and T. N. Thinh, "An anomaly-based intrusion detection architecture integrated on openflow switch," in *Proceedings of the 6th international conference on communication and network security*, 2016, pp. 99–103, doi: 10.1145/3017971.3017982.

[96] K. Muthamil Sudar and P. Deepalakshmi, "An intelligent flow-based and signature-based ids for sdns using ensemble feature selection and a multi-layer machine learning-based classifier," *Journal of Intelligent & Fuzzy Systems*, vol. 40, no. 3, pp. 4237–4256, 2021, doi: 10.3233/JIFS-200850.

[97] P. Krishnan, S. Duttagupta, and K. Achuthan, "Varman: Multi-plane security framework for software defined networks," *Computer communications*, vol. 148, pp. 215–239, 2019, doi: 10.1016/j.comcom.2019.09.014.

[98] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, 2016, pp. 258–263, doi: 10.1109/WINCOM.2016.7777224.

[99] "Deep recurrent neural network for intrusion detection in sdn-based networks," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018, pp. 202–206, doi: 10.1109/NETSOFT.2018.8460090.

[100] A. O. Alzahrani and M. J. Alenazi, "Designing a network intrusion detection system based on machine learning for software defined networks," *Future Internet*, vol. 13, no. 5, p. 111, 2021, doi: 10.3390/fi13050111.

[101] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information processing & management*, vol. 45, no. 4, pp. 427–437, 2009, doi: 10.1016/j.ipm.2009.03.002.

[102] A. Tharwat, "Classification assessment methods," *Applied computing and informatics*, vol. 17, no. 1, pp. 168–192, 2021, doi: 10.1016/j.aci.2018.08.003.

[103] J. Cui, J. Zhang, J. He, H. Zhong, and Y. Lu, "Ddos detection and defense mechanism for sdn controllers with k-means," in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, Leicester, UK, 2020, pp. 394–401, doi: 10.1109/UCC48980.2020.00062.

[104] I. Sumantra and S. I. Gandhi, "Ddos attack detection and mitigation in software defined networks," in *2020 International Conference on System, Computation, Automation and Networking (ICSCAN)*, Pondicherry, India, 2020, pp. 1–5, doi: 10.1109/ICSCAN49426.2020.9262408.

[105] B. Pfaff, B. Lantz, B. Heller, C. Barker, C. Beckmann, D. Cohn *et al.*, "Openflow switch specification," OPEN NETWORKING FOUNDATION, Tech. Rep. Version 1.3.0, 2012, accessed: 10 December 2023. [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf

[106] D. Arivudainambi, V. K. K.A, and S. S. Chakkaravarthy, "Lion ids: A meta-heuristics approach to detect ddos attacks against software-defined networks," *Neural Comput & Applic*, vol. 31, no. 5, pp. 1491–1501, May 2019, doi: 10.1007/s00521-018-3383-7.

[107] J. Ye, X. Cheng, J. Zhu, L. Feng, and L. Song, "A ddos attack detection method based on svm in software defined network," *Security and Communication Networks*, vol. 2018, pp. 1–8, 2018, doi: 10.1155/2018/9804061.

[108] F. Musumeci, A. C. Fidanci, F. Paolucci, F. Cugini, and M. Tornatore, "Machine-learning-enabled ddos attacks detection in p4 programmable networks," *J Netw Syst Manage*, vol. 30, no. 1, p. 21, Jan 2022, doi: 10.1007/s10922-021-09633-5.

[109] O. Kasim, "A robust dns flood attack detection with a hybrid deeper learning model," *Computers and Electrical Engineering*, vol. 100, p. 107883, May 2022, doi: 10.1016/j.compeleceng.2022.107883.

[110] K. M. Sudar and P. Deepalakshmi, "A two level security mechanism to detect a ddos flooding attack in software-defined networks using entropy-based and c4.5 technique," *JHS*, vol. 26, no. 1, pp. 55–76, Mar 2020, doi: 10.3233/JHS-200630.

[111] M. Ahmed, R. Seraj, and S. M. S. Islam, "The k-means algorithm: A comprehensive survey and performance evaluation," *Electronics*, vol. 9, no. 8, p. 1295, 2020, doi: 10.3390/electronics9081295.

[112] T. M. Ghazal, "Performances of k-means clustering algorithm with different distance metrics," *Intelligent Automation & Soft Computing*, vol. 30, no. 2, pp. 735–742, 2021, doi: 10.32604/iasc.2021.019067.

[113] H. Yzzogh, "Insdn_ddos_exp.rar," GitHub, 2024, accessed: 10 November 2023. [Online]. Available: https://github.com/Yzzogh/DDoS/blob/main/InSDN_DDoS_Exp.rar

[114] "Cic_ddos2017_exp.rar," GitHub, 2024, accessed: 10 November 2023. [Online]. Available: https://github.com/Yzzogh/DDoS/blob/main/CIC_DDoS2017_Exp.rar

[115] S. Wang, J. F. Balarezo, K. G. Chavez, A. Al-Hourani, S. Kandeepan, M. R. Asghar, and G. Russello, "Detecting flooding ddos attacks in software defined networks using supervised learning techniques," *Engineering science and technology, an international journal*, vol. 35, p. 101176, 2022, doi: 10.1016/j.jestch.2022.101176.

[116] A. O. Sangodoyin, M. O. Akinsolu, P. Pillai, and V. Grout, "Detection and classification of ddos flooding attacks on software-defined networks: A case study for the application of machine learning," *IEEE Access*, vol. 9, pp. 122 495–122 508, 2021, doi: 10.1109/ACCESS.2021.3109490.

[117] S. Y. Khamaiseh, A. Al-Alaj, and A. Warner, "Flooddetector: Detecting unknown dos flooding attacks in sdn," in *2020 International Conference on Internet of Things and Intelligent Applications (ITIA)*. IEEE, 2020, pp. 1–5, doi: 10.1109/ITIA50152.2020.9312310.

[118] K. S. Sahoo, B. K. Tripathy, K. Naik, S. Ramasubbareddy, B. Balusamy, M. Khari, and D. Burgos, "An evolutionary svm model for ddos attack detection in software defined networks," *IEEE access*, vol. 8, pp. 132 502–132 513, 2020, doi: 10.1109/ACCESS.2020.3009733.

[119] L. Zheng, J. Jiang, W. Pan, and H. Liu, "High-performance and range-supported packet classification algorithm for network security systems in sdn," in *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2020, pp. 1–6, doi: 10.1109/ICCWorkshops49005.2020.9145461.

[120] T. Jafarian, M. Masdari, A. Ghaffari, and K. Majidzadeh, "Sadm-sdnc: security anomaly detection and mitigation in software-defined networking using c-support vector classification," *Computing*, vol. 103, no. 4, pp. 641–673, 2021, doi: 10.1007/s00607-020-00866-x.

[121] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, M. Ghogho, and F. El Moussa, "Deepids: Deep learning approach for intrusion detection in software defined networking," *Electronics*, vol. 9, no. 9, p. 1533, 2020, doi: 10.3390/electronics9091533.

[122] N. S. Shaji, T. Jain, R. Muthalagu, and P. M. Pawar, "Deep-discovery: Anomaly discovery in software-defined networks using artificial neural networks," *Computers & Security*, vol. 132, p. 103320, 2023, doi: 10.1016/j.cose.2023.103320.

[123] J. van Staden and D. Brown, "An evaluation of machine learning methods for classifying bot traffic in software defined networks," in *Proceedings of Third International Conference on Sustainable Expert Systems: ICSES 2022*. Springer, 2023, pp. 979–991, doi: 10.1007/978-981-19-7874-6_72.

[124] L. Tan, Y. Pan, J. Wu, J. Zhou, H. Jiang, and Y. Deng, "A new framework for ddos attack detection and defense in sdn environment," *IEEE access*, vol. 8, pp. 161 908–161 919, 2020, doi: 10.1109/ACCESS.2020.3021435.

[125] Y. Xu, H. Sun, F. Xiang, and Z. Sun, "Efficient ddos detection based on k-fknn in software defined networks," *IEEE Access*, vol. 7, pp. 160 536–160 545, 2019, doi: 10.1109/ACCESS.2019.2950945.

[126] J. S. Rojas, A. R. Gallón, and J. C. Corrales, "Personalized service degradation policies on ott applications based on the consumption behavior of users," in *Computational Science and Its Applications–ICCSA 2018: 18th International Conference, Melbourne, VIC, Australia, July 2–5, 2018, Proceedings, Part III 18*. Springer, 2018, pp. 543–557, doi: 10.1007/978-3-319-95168-3_37.

[127] A. Madhukar and C. Williamson, "A longitudinal study of p2p traffic classification," in *14th IEEE International Symposium on Modeling, Analysis, and Simulation*, 2006, pp. 179–188, doi: 10.1109/MASCOTS.2006.6.

[128] M. Qin, K. Lei, B. Bai, and G. Zhang, "Towards a profiling view for unsupervised traffic classification by exploring the statistic features and link patterns," in *Proceedings of the 2019 Workshop on Network Meets AI & ML*, 2019, pp. 50–56, doi: 10.1145/3341216.3342213.

[129] E. Areström and N. Carlsson, "Early online classification of encrypted traffic streams using multifractal features," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019, pp. 84–89, doi: 10.1109/INFCOMW.2019.8845127.

[130] K. W. Church, "Word2vec," *Natural Language Engineering*, vol. 23, no. 1, pp. 155–162, 2017, doi: 10.1017/S1351324916000334.

[131] D. E. Cahyani and I. Patasik, "Performance comparison of tf-idf and word2vec models for emotion text classification," *Bulletin of Electrical Engineering and Informatics*, vol. 10, no. 5, pp. 2780–2788, 2021, doi: 10.11591/eei.v10i5.3157.

[132] H. Yzzogh, "InSDN_Traffic_Analysis.rar," GitHub, 2024, accessed: 10 November 2023. [Online]. Available: https://github.com/Yzzogh/Traffic-Analysis/blob/main

[133] "CIC-DDoS2019_Traffic_Analysis.rar," GitHub, 2024, accessed: 10 November 2023. [Online]. Available: https://github.com/Yzzogh/Traffic-Analysis/blob/main

[134] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[135] G. Rao, W. Huang, Z. Feng, and Q. Cong, "Lstm with sentence representations for document-level sentiment classification," *Neurocomputing*, vol. 308, pp. 49–57, 2018, doi: 10.1016/j.neucom.2018.04.045.

[136] A. Malik, R. de Fréin, M. Al-Zeyadi, and J. Andreu-Perez, "Intelligent sdn traffic classification using deep learning: Deep-sdn," in *2020 2nd International Conference on Computer Communication and the Internet (ICCCI)*, 2020, pp. 184–189, doi: 10.1109/ICCCI49374.2020.9145971.

[137] D. Nuñez-Agurto, W. Fuertes, L. Marrone, E. Benavides-Astudillo, C. Coronel-Guerrero, and F. Perez, "A novel traffic classification approach by employing deep learning on software-defined networking," *Future Internet*, vol. 16, no. 5, p. 153, 2024, doi: 10.3390/fi16050153.

[138] N. Ahuja, G. Singal, and D. Mukhopadhyay, "Dlsdn: Deep learning for ddos attack detection in software defined networking," in *2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, 2021, pp. 683–688, doi: 10.1109/Confluence51648.2021.9376879.

[139] L. Chen, S. Li, Q. Bai, J. Yang, S. Jiang, and Y. Miao, "Review of image classification algorithms based on convolutional neural networks," *Remote Sensing*, vol. 13, no. 22, p. 4712, 2021, doi: 10.3390/rs13224712.

[140] S. Kala, D. Paul, B. R. Jose, J. Mathew, and S. Nalesh, "Performance analysis of convolutional neural network models," in *2019 9th International Conference on Advances in Computing and Communication (ICACC)*. IEEE, 2019, pp. 22–26, doi: 10.1109/ICACC48162.2019.8986201.

[141] L. C. Das, "Traffic volume prediction using memory-based recurrent neural networks: A comparative analysis of lstm and gru," *arXiv preprint arXiv:2303.12643*, 2023, doi: 10.48550/arXiv.2303.12643.

[142] N. M. Rezk, M. Purnaprajna, T. Nordström, and Z. Ul-Abdin, "Recurrent neural networks: An embedded computing perspective," *IEEE Access*, vol. 8, pp. 57 967–57 996, 2020, doi: 10.1109/ACCESS.2020.2982416.

[143] M. R. Hadi and A. S. Mohammed, "A novel approach to network intrusion detection system using deep learning for sdn: Futuristic approach," *arXiv preprint arXiv:2208.02094*, 2022, doi: 10.5121/csit.2022.121106.

[144] Y. Zhu, T. Brettin, F. Xia, A. Partin, M. Shukla, H. Yoo, Y. A. Evrard, J. H. Doroshow, and R. L. Stevens, "Converting tabular data into images for deep learning with convolutional neural networks," *Scientific reports*, vol. 11, no. 1, p. 11325, 2021, doi: 10.1038/s41598-021-90923-y.

[145] T. Pinthong, W. Yimyam, N. Chumuang, M. Ketcham, P. Pramkeaw, and N. Utakrit, "Image classification of forage plants in fabaceae family using scale invariant feature transform method," in *2020 15th International Joint Symposium on Artificial Intelligence and Natural Language Processing (iSAI-NLP)*, 2020, pp. 1–6, doi: 10.1109/iSAI-NLP51646.2020.9376824.

[146] K. S. M. Shabbir, M. I. Ahmed, and M. Alam, "Detection of glaucoma using orb (oriented fast and rotated brief) feature extraction," *Journal of Engineering Advancements*, vol. 2, no. 03, pp. 153–158, 2021, doi: 10.38032/jea.2021.03.005.

[147] D. G. Viswanathan, "Features from accelerated segment test (fast)," in *Proceedings of the 10th workshop on image analysis for multimedia interactive services, London, UK*, 2009, pp. 6–8.

[148] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11*. Springer, 2010, pp. 778–792, doi: 10.1007/978-3-642-15561-1_56.

[149] H. Yzzogh, "Insdn-trafficclassification.csv," GitHub, 2025, accessed: 6 January 2025. [Online]. Available: https://github.com/Yzzogh/Network-Traffic-Classification-Dataset/blob/main/InSDN-TrafficClassification.csv

[150] "Cic-ddos2019-trafficclassification.csv," GitHub, 2025, accessed: 8 January 2025. [Online]. Available: https://github.com/Yzzogh/Network-Traffic-Classification-Dataset/blob/main/CIC-DDoS2019-TrafficClassification.csv

[151] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information processing & management*, vol. 45, no. 4, pp. 427–437, 2009, doi: 10.5121/csit.2022.121106.

[152] A. Tharwat, "Classification assessment methods," *Applied computing and informatics*, vol. 17, no. 1, pp. 168–192, 2021, doi: 10.1016/j.aci.2018.08.003.

[153] S. A. K. Tareen and Z. Saleem, "A comparative analysis of sift, surf, kaze, akaze, orb, and brisk," in *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, 2018, pp. 1–10, doi: 10.1109/ICOMET.2018.8346440.

[154] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems*, vol. 33, no. 12, pp. 6999–7019, 2021, doi: 10.1109/TNNLS.2021.3084827.

[155] A. Canclini, M. Cesana, A. Redondi, M. Tagliasacchi, J. Ascenso, and R. Cilla, "Evaluation of low-complexity visual feature detectors and descriptors," in *2013 18th International Conference on Digital Signal Processing (DSP)*, 2013, pp. 1–7, doi: 10.1109/ICDSP.2013.6622757.

[156] B. Pfaff, "Openflow switch specification," 2012. [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf

[157] R. Swami, M. Dave, and V. Ranga, "Detection and analysis of tcp-syn ddos attack in software-defined networking," *Wireless Personal Communications*, vol. 118, no. 4, pp. 2295–2317, Jun 2021, doi: 10.1007/s11277-021-08127-6.

[158] N. Ahuja, G. Singal, and D. Mukhopadhyay, "Dlsdn: Deep learning for ddos attack detection in software defined networking," in *2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, 2021, pp. 683–688, doi: 10.1109/Confluence51648.2021.9376879.

[159] P. Choobdar, M. Naderan, and M. Naderan, "Detection and multi-class classification of intrusion in software defined networks using stacked auto-encoders and cicids2017 dataset," *Wireless Personal Communications*, vol. 123, no. 1, pp. 437–471, 2022, doi: 10.1007/s11277-021-09139-y.

[160] D. Nuñez-Agurto, W. Fuertes, L. Marrone, E. Benavides-Astudillo, C. Coronel-Guerrero, and F. Perez, "A novel traffic classification approach by employing deep learning on software-defined networking," *Future Internet*, vol. 16, no. 5, p. 153, 2024, doi: 10.3390/fi16050153.

[161] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related," in *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, 2016, pp. 407–414, doi: 10.5220/0005740704070414.

[162] S. V. Margariti, I. G. Tsoulos, E. Kiousi, and E. Stergiou, "Traffic classification in software-defined networking using genetic programming tools," *Future Internet*, vol. 16, no. 9, p. 338, 2024, doi: 10.3390/fi16090338.

Abstract

The rapid evolution of digital services and applications has increased the complexity of modern networks. Software-Defined Networks (SDN) offer centralized management and enhanced programmability by separating control from the data plane. However, the centralization of the control plane exposes SDNs to DDoS attacks and other advanced threats, compromising the reliability of services. This thesis explores the application of Machine Learning to enhance SDN security through traffic classification and intrusion detection. Traditional methods, such as port-based filtering and deep packet inspection, are inadequate for the high volume, encrypted traffic, and dynamic nature of modern networks. Machine learning enables more accurate identification of emerging cyber threats. Three approaches are studied: the first combines K-Means clustering with Naïve Bayes to detect DDoS flood attacks. The other two focus on traffic classification: one combines K-Means and Word2Vec with a neural network, while the other applies feature extraction using Oriented FAST and Rotated BRIEF (ORB), followed by neural network-based classification. These methods strengthen SDN security by offering innovative solutions.

Keywords (10)*: Software-Defined Networking, Machine Learning, Security, Flooding DDoS Attacks, K-Means, Naïve Bayes, Word2Vec, Neural Network, Oriented FAST and Rotated BRIEF, Traffic Classification.*

Résumé

L'évolution rapide des services et des applications numériques a augmenté la complexité des réseaux modernes. Les réseaux définis par logiciel (SDN) offrent une gestion centralisée et une programmabilité accrue en séparant le contrôle du transfert de données. Cependant, la centralisation du plan de contrôle expose les SDN aux attaques DDoS et à d'autres menaces avancées, compromettant ainsi la fiabilité des services. Cette thèse explore l'application du Machine Learning pour améliorer la sécurité des SDN via la classification du trafic et la détection d'intrusions. Les méthodes traditionnelles, comme le filtrage par port et l'inspection de paquets, sont inadéquates face au volume élevé, au trafic chiffré et à la nature dynamique des réseaux modernes. L'apprentissage automatique permet une identification plus précise des cybermenaces émergentes. Trois approches sont étudiées : la première combine le clustering K-Means avec Naïve Bayes pour détecter les attaques DDoS par inondation. Les deux autres concernent la classification du trafic : l'une combine K-Means et Word2Vec avec un réseau de neurones, tandis que l'autre applique l'extraction de caractéristiques avec Oriented FAST and Rotated BRIEF (ORB), suivie d'une classification par réseau de neurones.

Mots-clés (10) *: Réseaux définis par logiciel, Apprentissage automatique, Sécurité, Attaques DDoS par inondation, K-Means, Naïve Bayes, Word2Vec, Réseau de neurones, Oriented FAST and Rotated BRIEF, Classification du trafic.*