SELF-HEALING AND LIGHTWEIGHT INTRUSION DETECTION FOR IOT-BASED SYSTEMS

MAHAWISH

02-200182-006

A thesis submitted in fulfillment of the
requirements for the award of the degree of
Doctor of Philosophy (Software Engineering)

DEPARTMENT OF SOFTWARE ENGINEERING

BAHRIA UNIVERSITY, KARACHI

DECEMBER 2025

Approval for Examination

Scholar's Name: Mahawish Registration No.: 02-200182-006 Program of Study: Doctor of Philosophy (Software Engineering) Thesis Title: "Self-healing and Lightweight Intrusion Detection for IoT based Systems."

It is to certify that the above scholar's thesis has been completed to my satisfaction and that its standard is appropriate for submission for examination. I have also conducted a plagiarism test of this thesis using HEC-prescribed software and found a similarity index 13 % within the permissible limit set by the HEC for the PhD degree thesis. I have also found the thesis in a format recognized by the BU for the PhD thesis.

Principal Supervisor's Signature:  Date: 30 DECEMBER 2025

Name: Dr. Osama Rehman

Author's Declaration

I, Mahawish hereby state that my PhD thesis titled "Self-healing and Lightweight Intrusion Detection for IoT based Systems " is my own work and has not been submitted previously by me for taking any degree from this university Bahria University or anywhere else in the country/world.

At any time if my statement is found to be incorrect even after my graduation, the University has the right to withdraw/cancel my PhD degree.

Name of scholar: Mahawish

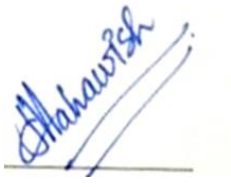
Date: 30 DECEMBER 2025

Plagiarism Undertaking

I, solemnly declare that the research work presented in the thesis titled "Self-healing and Lightweight Intrusion Detection for IoT based Systems." is solely my research work with no significant contribution from any other person. Small contribution/help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero tolerance policy of the HEC and Bahria University towards plagiarism. Therefore I as an Author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of PhD degree, the university reserves the right to withdraw/revoke my PhD degree and that HEC and the University has the right to publish my name on the HEC/University website on which names of scholars are placed who submitted plagiarized thesis.

Scholar/Author's Sign: 

Name of the Scholar: Mahawish

Acknowledgment

In the name of Allah, the Most Gracious and the Most Merciful

All praises to Allah, He gave me the strength and courage to complete this thesis. Pursuing a Ph.D. degree is a great learning experience and it would not have been possible without the support of many people. First and foremost, I am extremely thankful to my supervisor Dr. Osama Rehman. His encouraging words and detailed feedback have been very important to me. He welcomed me as his Ph.D. student despite his assiduous and busy life. His brilliant, skillful supervision enriched this study higher than my expectations.

I would also like to thank Bahria University (Karachi Campus) for providing me with a perfect environment and resources whenever needed which made the whole process a lot easier.

Most importantly, I am grateful for my friends and family's unconditional, unequivocal, and loving support throughout the entire thesis process and every day. They are the most valuable people in my life and I dedicate this dissertation to them.

Abstract

The rapid growth of Internet of Things (IoT) has revolutionized modern technology, enabling seamless connectivity across diverse domains. However, resource-constrained IoT devices are highly vulnerable to cyber threats, particularly Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks. Such attacks can severely compromise the reliability and functionality of IoT networks, necessitating robust, resource-efficient, and adaptive intrusion detection mechanisms. Traditional Intrusion Detection Systems (IDSs) often fail to meet the unique demands of IoT environments due to their high computational overhead and inability to adapt dynamically to emerging threats. Addressing these challenges requires an innovative approach that balances lightweight design with high detection accuracy and resilience. The first part of this research proposes an Ensemble Lightweight Intrusion Detection System (ELIDS) approach, which combines the capabilities of seven distinct filter-based Feature Selection (FS) methods to identify the most relevant features in classifying normal and DoS/DDoS attack packets. In the proposed ELIDS, each FS method generates a ranked list of features along with their corresponding scores. These lists are subsequently aggregated, resulting in a consolidated final list that includes the reduced set of features. The selected features are further utilized to train six Machine Learning (ML) algorithms, designing lightweight ML-enabled IDS. The proposed ELIDS is evaluated for in-domain and cross-domain testing scenarios. The results reveal that incorporating an ensemble FS approach improves detection accuracy also optimizes computational resources. Moreover, ELIDS outperforms conventional FS methods, especially over cross-domain testing scenarios. The second part of this research presents a Self-Healing for Internet of Thing (SH-

IoT), designed to enhance the performance of ML-enabled IDS models, especially when a decline in their effectiveness is observed during cross-domain testing. In this framework, the health monitor continuously checks the behavior of simulated IoT devices and generates a danger signal when it detects any deviation from normal operating patterns caused by a DoS/DDoS attack. Once the danger signal is triggered, the proposed SHIoT initiates its healing mechanism, which activates defensive measures to counter DoS/DDoS attacks. The framework is assessed using performance metrics and obtained results demonstrate a notable enhancement in the performance of ML models, which retain both previous and newly acquired knowledge to identify both existing and emerging DoS/DDoS attacks. Finally, the third part of this research integrates the Self-Healing and Ensemble-based Lightweight IDS (SHIoT-ELIDS). The SHIoT-ELIDS is evaluated using nine unseen datasets, achieving a peak accuracy of 99.9% while employing Random Forest as the learning algorithm. Additionally, the proposed SHIoT-ELIDS performs efficiently in terms of resource utilization, where testing a single packet takes only 0.0050 msec, demonstrating its appropriateness for real-time detection environments. The CPU usage is around 0.0014% per packet, ensuring the system imposes minimal load on the underlying hardware. Additionally, memory consumption remains low at 0.1456 MB per packet, while the final classification model size is of 697 KB, making it ideal for deployment on devices with limited processing and storage capacity, such as IoT devices. Overall, SHIoT-ELIDS can improve the security posture of IoT networks, providing a scalable and resource-efficient defense against evolving cyber threats.

Table of Contents

List of Publications

Fatima, M., Rehman, O., Ali, S., & Niazi, M. F. (2024). ELIDS: Ensemble Feature Selection for Lightweight IDS against DDoS Attacks in Resource-Constrained IoT Environment. Future Generation Computer Systems, 159, 172-187.

Fatima, M., Rehman, O., Rahman, I. M., Ajmal, A., & Park, S. J. (2024). Towards Ensemble Feature Selection for Lightweight Intrusion Detection in Resource-Constrained IoT Devices. Future Internet, 16(10).

Fatima, M., Rehman, O., & Rahman, I. M. (2024, August). Ensemble Feature Selection based Lightweight IDS Tailored for DDoS Attacks Detection over IoT Devices. In 2024 International Visualization, Informatics and Technology Conference (IVIT) (pp. 20-27). IEEE.

Fatima, M., Rehman, O., & Rehman, I. M. (2023, July). Li-IDS: An Approach Towards a Lightweight IDS for Resource-Constrained IoT. In 2023 International Conference on Smart Applications, Communications and Networking (SmartNets) (pp. 1-6). IEEE.

Fatima, M., Rehman, O., N. Z. Jhanjhi, & Saqib Ali. SH-IDS: A Resilient Self-Healing Intrusion Detection Framework Against DoS and DDoS Attacks in IoT Systems. Scientific reports [**Submitted**]

List of Figures

List of Tables

List of Abbreviations

IoT	Internet of Things
IDS	Intrusion Detection System
NIDS	Network Intrusion Detection System
HIDS	Host-based Intrusion Detection System
ML	Machine Learning
ELIDS	Ensemble Lighweight Intrusion Detection System
SHIoT	Self-Healing for IoT
AI	Artificial Intelligence
RF	Random Forest
NN	Neural Network
GB	Gradient Boosting
NB	Naive Bayes
KNN	K-Nearest Neighbor
DT	Decision Tree
FPR	False Positive Rate
TPR	True Positive Rate
TNR	True Negative Rate
FNR	False Negative Rate
ms	milliseconds
MB	Megabytes

FS	Feature Selection
MI	Mutual Information
VT	Variance Threshold
PCC	Pearson's Correlation Coefficient
MAD	Mean Absolute Difference
χ^2	Chi-Square
DR	Dispersion Ratio
F-Score	Fisher's Score
MitM	Man-in-the-Middle
DoS	Daniel of Service
DDoS	Distributed Daniel of Service
ICMP	Internet Control Message Protocol
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
DNN	Deep Neural Network
HTTP	Hyper Text Transfer Protocol
DNS	Domain Name System
AWS	Amazon Web Services
PCA	Principal Component Analysis
MLP	Multi-Layer Perceptron
QoS	Quality of Service
AMS	Alert Management System
IMS	Intrusion Mitigation System
pps	Packet Per Second
EDAS	Evaluation Based on Distance from Average Solution
MCDM	Multiple Criteria Decision-Making Problem

ACO	Ant Colony Optimization
PSO	Particle Swarm Optimization
GA	Genetic Algorithms
LSTM	Long Short Term Memory
SenS	Sensing Approach
ReAct	Responsibility and Actions
MAPE-K	Monitor, Analyze, Plan, Execute, Knowledge
AIS	Artificial Immune System

CHAPTER 1

Introduction

The Internet of Things (IoT) represents a sophisticated paradigm that has significantly influenced contemporary computing. This network encompasses a wide array of devices, including sensors and actuators, which autonomously send and receive data over the Internet or other communication networks, depending on the application context. The IoT has revolutionized international interaction and communications as a result of seamless connectivity for billions of devices, which in essence lessens human interaction [1, 2]. The paradigm shift brought about by IoT provides tremendous advantages for distant environmental observation and control, improved decision-making mechanisms, and ultimately improves the quality of life for people as a result of smart operation [3].

The growth of IoT devices keeps increasing with predictions indicating a projected number of 50 billion in 2030 [4]. In addition, the IoT is expected to have a great economic impact with estimates indicating a global value of between 10.5

trillion USD to 12.6 trillion USD in 2030 [5].

This network of sensors and actuators significantly changes the way one lives by enabling reliable connectivity using the seamless fusion of various devices and systems. This trend, as depicted in Figure 1.1, ranges across various domains like smart homes, industries, healthcare, agriculture, transport systems, and intelligent logistics.

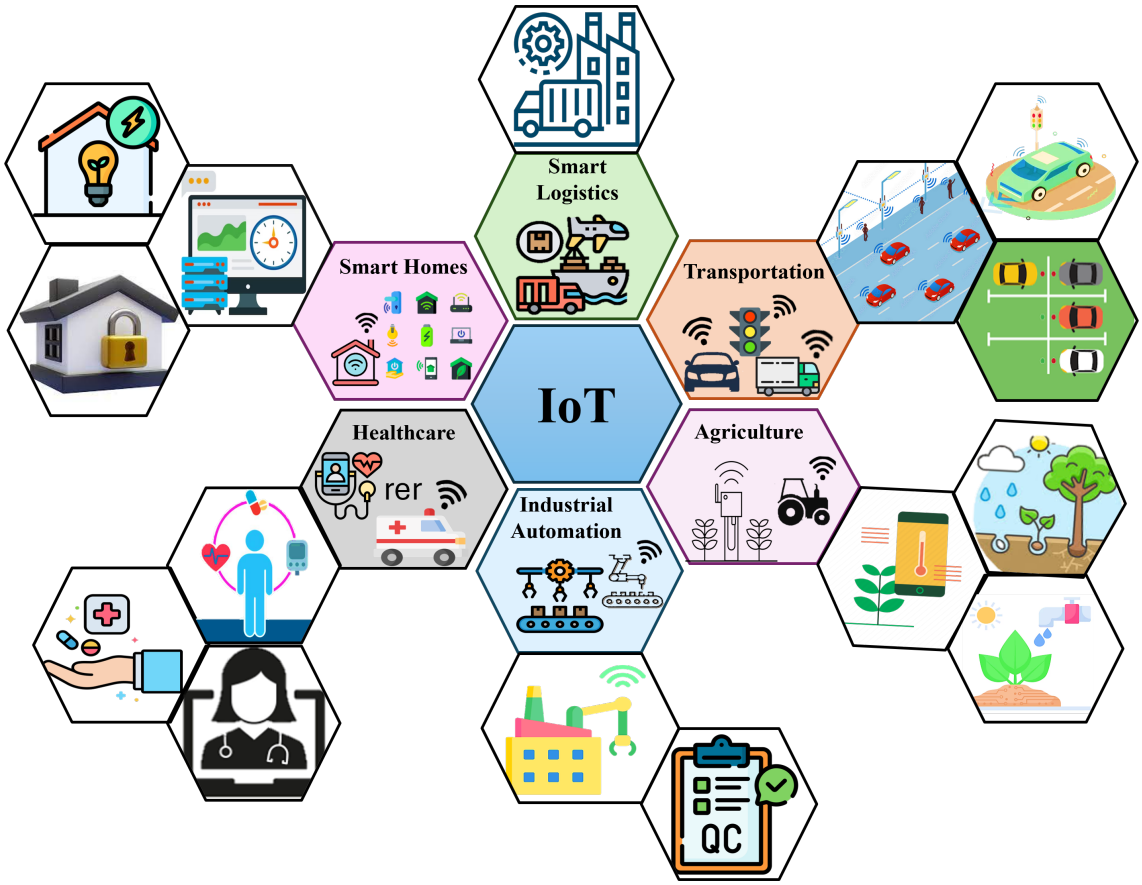


Figure 1.1: Application domains in IoT systems.

The Internet of Medical Things (IoMT) helps monitor patients continuously and perform remote and timely medical procedures [6]. In the home. IoT helps enhance home security and remote monitoring. For industrial automation purposes,

IoT helps optimize procedures like monitoring the quality of goods and material management. For transportation logistics, the Internet of Vehicles (IoV) helps achieve car-to-car and car-to-infrastructure communication that provides monitoring of road traffic, parking systems, and optimized routes. Furthermore, the IoT helps achieve a higher level of inventory management and improved customers' satisfaction services that optimize the delivery of goods and services [7].

1.1 Smart Devices and Vulnerabilities

IoT systems comprise smart devices embedded with sensors, actuators, and communication interfaces, allowing them to collect, process, and transmit data in real-time [8]. Figure 1.2 provides an illustrative diagram of a typical IoT ecosystem that presents the typical components commonly found in an IoT system. The monitored environment/object represents the physical environment that is to be monitored, such as an agricultural field, manufacturing plant, power grid generators, home appliances, and human beings. The IoT devices that are usually deployed near to the monitored environment/object can be either classified as IoT end devices or IoT gateways [9]. IoT end devices are represented in Figure 1.2, usually hosts the sensors, processing board and a low-power communication module. While the IoT gateways, also known as hubs, act as aggregators for the data generated by the IoT sensors and also provide a first level of data connectivity between the sensing nodes and rest of the network. Both IoT end devices and IoT gateways are generally known to be resource-constrained devices within the IoT ecosystem as opposed to the conventional IT networks in which such devices are known to have an abundant

amount of resources.

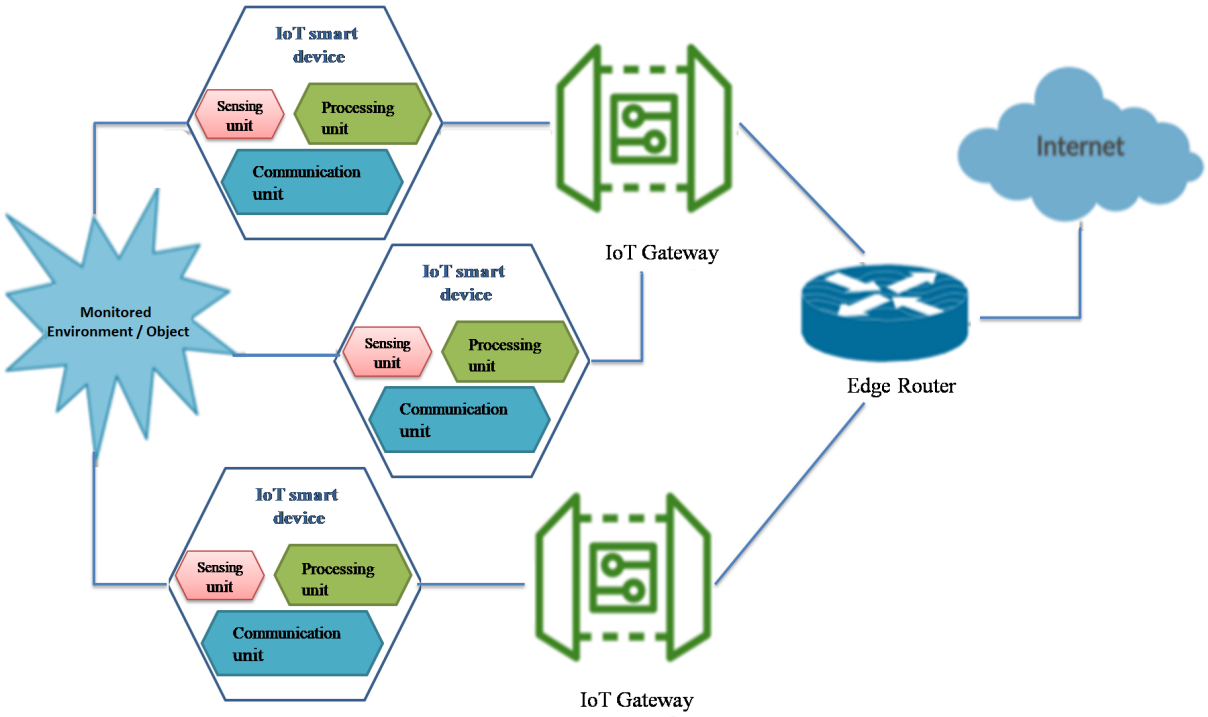


Figure 1.2: General architecture of IoT systems.

IoT devices provide substantial benefits by automating various aspects of everyday life and enhancing convenience, efficiency, and functionality. However, these devices are prone to serious security threats because of numerous inherent vulnerabilities. As highlighted by [10], these vulnerabilities include weak passwords, backdoors, software flaws, ineffective software update policies, insecure web and mobile interfaces, insufficient encryption and authorization mechanisms, and inadequate physical security measures.

A major reason behind the security risks associated with IoT devices is their resource constraints on computing capabilities. Most IoT devices are characterized by their restricted processing capabilities and limited memory, thus preventing the

adoption of effective security protocols [11]. To give an example, the use of encryption protocols requires high processing capabilities for efficient execution of encryption and decryption processes, with resource-constrained devices potentially being unable to support such operations effectively and thus preventing the security of the data being realized. Also, the use of software updates, being an effective means for improving security, could be threatened due to resource constraints. Furthermore, effective password protection requires secure processing and thus may be limited by resource constraints on IoT devices, with secure storage and processing mandating high resource capabilities and memory capacity. Moreover, anomaly-based IDS that aim at identifying threats by observing any inconsistencies with normal network traffic behavior and not specific signatures continue to demand high resource capabilities on the part of the device for effective continuous observation and subsequent real-time reaction for potential security threats [12].

1.2 DoS/DDoS Attacks in IoT

A Denial of Service (DoS) attack floods the target system with an enormous number of packets that contain fraudulent requests, which causes the system to become occupied with handling such illegitimate traffic and fail to address the genuine user service requests [13].

In a similar fashion, a Distributed Denial of Service (DDoS) attack also leverages a massive number of compromised devices for the purpose of overwhelming a target system/network. The attacking individual/s controls the devices and instruct them on how to send a massive amount of fake requests towards the target resulting

in the inability to differentiate between genuine and fake traffic.

IoT devices are intended to be resource-constrained in terms of CPU and power consumption to fulfill particular tasks. Examples of resource-constrained devices include smart thermostats, security cameras, and health monitors. Because of these constraints, the vulnerability of IoT devices to DoS/DDoS attacks increases to a larger extent. In a DoS/DDoS attack, it has been noticed that the targeted IoT devices might become non-functional and enter a state of abnormal or dysfunctional behavior [14]. For example, a temperature-control system might not be capable of controlling the temperature in the desired range; a security camera might neither record nor broadcast; and a health monitor might generate invalid readings or become totally non-functional [15].

The following examples show the increasing frequency and sophistication of DoS/DDoS attacks specifically targeting IoT devices:

- **Mirai:** A well-known botnet which made use of IoT (IoT) devices like IP cameras and routers to perform a massive DDoS attack and hence impacted well-known services like Twitter and Spotify [16].
- **Satori:** A Mirai botnet variant that exploits vulnerable Huawei routers to spread quickly and create a massive impact by disseminating information [17].
- **Mozi:** A botnet that was operational between late 2019 and 2021 and infected IoT devices with weak passwords that it used for DDoS attacks [18].
- **Hajime:** A peer-to-peer IoT botnet that showcased the capability of DDoS attacks on compromised devices like webcams and Digital Video Recorders

(DVRs) [19].

Intrusion Detection System (IDS) is a basic security system with the aim of analyzing and scanning network or device activities to identify signs of malicious and/or unauthorized usage. The main target of using IDS is to identify intrusion attempts capable of affecting the confidentiality, integrity, and availability of a computer system. There are three major types of intrusion detection techniques: signature-based techniques based on attack patterns; anomaly-based techniques utilizing abnormal activities to identify unknown or "zero-day" threats; and hybrid techniques combining anomaly and signature-based techniques to improve comprehensiveness and effectiveness. Such techniques are quite useful in typical computer networks but are associated with high overhead costs and are therefore unsuitable for use in typical IoT systems and devices.

With the wide implementation of IoT technology, maintaining the confidentiality, integrity, and availability of computer systems and networks has become a pressing issue. IoT technology possesses naturally vulnerable capabilities that attract attacks from malicious actors. To overcome the challenges associated with IoT technology attacks, the implementation of IDS acts as a crucial component that provides a safety net within the IoT domain. IDS systems are primarily designed to monitor the activities of IoT networks and trigger notifications on the onset of attack attempts on IoT technology domains, including the onset of DoS and DDoS attacks. An IDS recognizes both internal and external attacks on the safety of the IoT domain and offers IoT technology real-time protection capabilities that are naturally suited for IoT technology.

1.3 ML-based Security Solutions for IoT Systems

Various resource-efficient techniques are also being explored and developed to tackle the emerging security threats faced by resource-limited IoT networks, such as advanced cryptography techniques, secured firmware update techniques, and access control techniques [20]. These techniques are designed in a way to work within the limitations provided by the nature of IoT technology itself and strike a correct balance between consumed resources and provided security. However, they are also hampered by the constantly evolving environment of Internet threats [21].

1.3.1 ML-enabled Lightweight IDS

However, with the increasing threat levels, the focus is gradually shifting to the use of highly advanced technologies, with primary emphasis on Artificial Intelligence (AI). Among the AI technologies, the use of Machine Learning (ML) has appeared as an important tool for securing IoT networks [20]. It is possible to use ML for designing an efficient IDS capable of detecting and preventing attackers by learning their behavior [22]. Through their ability to learn from past experiences and track known threat patterns, the ML system is capable of detecting regular users and attackers with an adaptive security system [20].

Lightweight IDS secure the network by identifying malicious packets with less computational resource usage. ML-based lightweight IDS are normally designed by minimizing the features used in building the network traffic classification model [23]. Feature Selection (FS) is a technique that removes irrelevant features that are

redundant in a feature set [24]. A lightweight IDS based on ML that uses FS can function efficiently in identifying intrusion attacks in an IoT network with a low consumption of computational resources. Conventional resource-efficient methods build a sound foundation to secure IoT devices, but incorporating ML is a visionary approach to tackle the rising complexity of cyber threats. Such technologies offer superior detection and prevention mechanisms by continuously learning and adapting to new means of assault, and thereby making the overall security of IoT systems better.

1.3.2 Self-Healing Mechanism in IDS

Traditional IDS, which include statistical and expert-system strategies, protect against known traditional cyber threats through assessment of attack patterns. However, traditional IDSs often suffer from inefficiency during adaptive monitoring of evolving threats based on attack mechanisms. The self-healing process equips IDS with active mechanisms for detecting and healing from adverse or unsure conditions. Self-healing can make a system resilient to attack effects that enable normal system operation despite adversary impact [25]. With increasing use of sophisticated attack methodologies that can dynamically make use of system vulnerability, integration of self-healing functionalities into IDS is a fundamental need for future-proof security designs [25]. Through a survey research, [26] defined self-healing as a process ingredient that monitors any system behavior drift and initiates corrective operations accordingly. Self-healing applies adaptive strategies to enable attack recognition and subsequent recovery operations to incremental attack effects. The integration

of self-healing into an IDS can effectively deal with adverse effects of zero-day attacks through adaptive modification of a real-time intrusion attack model. Current research works, as presented in [27] and [28], emphasize that a self-healing IDS is extremely important and fundamental to ensuring IoT system integrity and dependability.

1.4 Problem Statement

Exiting ML-based IDS are resource-intensive to be deployed effectively over resource-constrained IoT devices. Moreover, existing ML-based IDS solutions often experience a decline in performance when dealing with emerging DoS/DDoS attacks, due to their inability to adapt to evolving threats and new techniques used to launch DoS/DDoS attacks. Hence, there is a critical need for a ML-based lightweight IDS that ensures robust detection of DoS/DDoS attacks without excessive resource consumption. In addition, there is a need for an IDS that incorporates self-healing capability to effectively counter both existing and emerging DoS/DDoS threats.

1.5 Research Objectives

Based on the literature review and preliminary analysis conducted, the research objectives are as follows:

1. To design and implement an ML-enabled lightweight IDS mechanism tailored to detect DoS/DDoS attacks within resource-constrained IoT devices.
2. To develop a self-healing mechanism that enables the IDS to autonomously

recover from state-of-the-art DoS/DDoS attacks.

3. To integrate the proposed lightweight and self-healing mechanisms to design an optimized IDS solution for IoT devices, ensuring resilience against emerging DoS/DDoS attacks.

1.6 Research Contributions

The research presents several significant contributions to the field of IoT security, particularly in developing a lightweight security solution tailored for resource-constrained IoT devices to protect against evolving DoS/DDoS attacks with self-healing capabilities. The major contributions of the research are:

- The study proposes a novel binary classification scheme that has been implemented in a novel Ensemble Lightweight Intrusion Detection System (ELIDS) to secure resource-intensive IoT networks from DoS/DDoS attacks. The proposed ensemble technique improves the robustness of the FS (FS) stage by combining the strengths of seven different filter-based FS techniques using a single framework. ELIDS refers to a major drawback in utilizing a solo technique that may result in biased results owing to the risk of overfitting a pattern in the training phase. Hence, there would be a risk of reduced accuracy in a real-world setup. The proposed ensemble FS technique eliminates this drawback, resulting in an optimal set of features that would enable training of six ML models for binary classification purposes. ELIDS has been thoroughly evaluated in a cross-domain setup that showcases the versatility of the pro-

posed technique in a broad spectrum of network scenarios.

- The second part of this work proposes a Self-Healing Internet of Things (SH-IoT) system that is intended for improving the efficiency of an intrusion detection system (IDS) in mitigating disruptions caused by newly emerging or sophisticated forms of DoS and DDoS attacks in a simulated atmosphere. Compared to traditional IDS solutions that cannot recover from disruptions themselves due to high levels of false positives and decreased accuracy in real-world implementations, the proposed SHIoT solution provides a dynamic and adaptive solution for improving IDS detection performance. It consists of a health monitor, where if it identifies resource depletion in IoT components, it triggers a danger signal. This danger signal starts the packet retraining process for updating the IDS model. The retraining process allows the IDS system to update its model with the most up-to-date data, resulting in improved detection efficiency against DoS/DDoS attacks.

- The last part of this work comprises the integration process of the Self-Healing and Ensemble Lightweight Intrusion Detection System (SHIoT-ELIDS), an advanced mechanism designed with the aim of addressing imminent DoS and DDoS attacks on the resource-poor IoT environment. Through this integration, a novel synthesis is achieved that considers the real constraints of the IoT device, such as its processing capabilities and the capacity for storing memory. SHIoT-ELIDS helps overcome limitations brought about by traditional ML algorithm-driven intrusion detection systems, such as high resource complexity and high false alarm rates that disruptively impact system func-

tionality.

1.7 Research Work Flow Diagram

Figure 1.3 below highlights a systematic research procedure for constructing a safe and resilient intrusion detection system (IDS) specifically designed for IoT scenarios. The procedure starts with a comprehensive literature study, which aids in the recognition of research gaps and formulation of three research objectives. The three research objectives support the research study in a distinct way: whereas Objective 1 helps in developing the ELIDS framework, Objective 2 assists in building the SHIoT framework, and finally, Objective 3 enables the integration of both into the SHIoT-ELIDS framework. Both frameworks undergo evaluation and result analysis, with a summary of the findings presented at the end of the respective chapter. The future scope is discussed in the final chapter.

1.8 Organisation of Thesis

The rest of the thesis is organized in the following chapters.

Chapter 2 It introduces the issues related to security in IoTsystems, including DoS and DDoS attacks in an IoT network, as well as the self-healing process. This literature also examines previous and current studies related to security solutions for IoT applications and studies about self-healing in IoT.

Chapter 3 Discussion of this chapter will focus on the ensemble FS method, which is a major part of the proposed system. It explains the criteria used in the

selection of the most important features based on the TON_IoT dataset using seven different filter-based FS techniques. An in-depth review of the contribution of the methods towards FS as well as their impact on the performance of the system is presented.

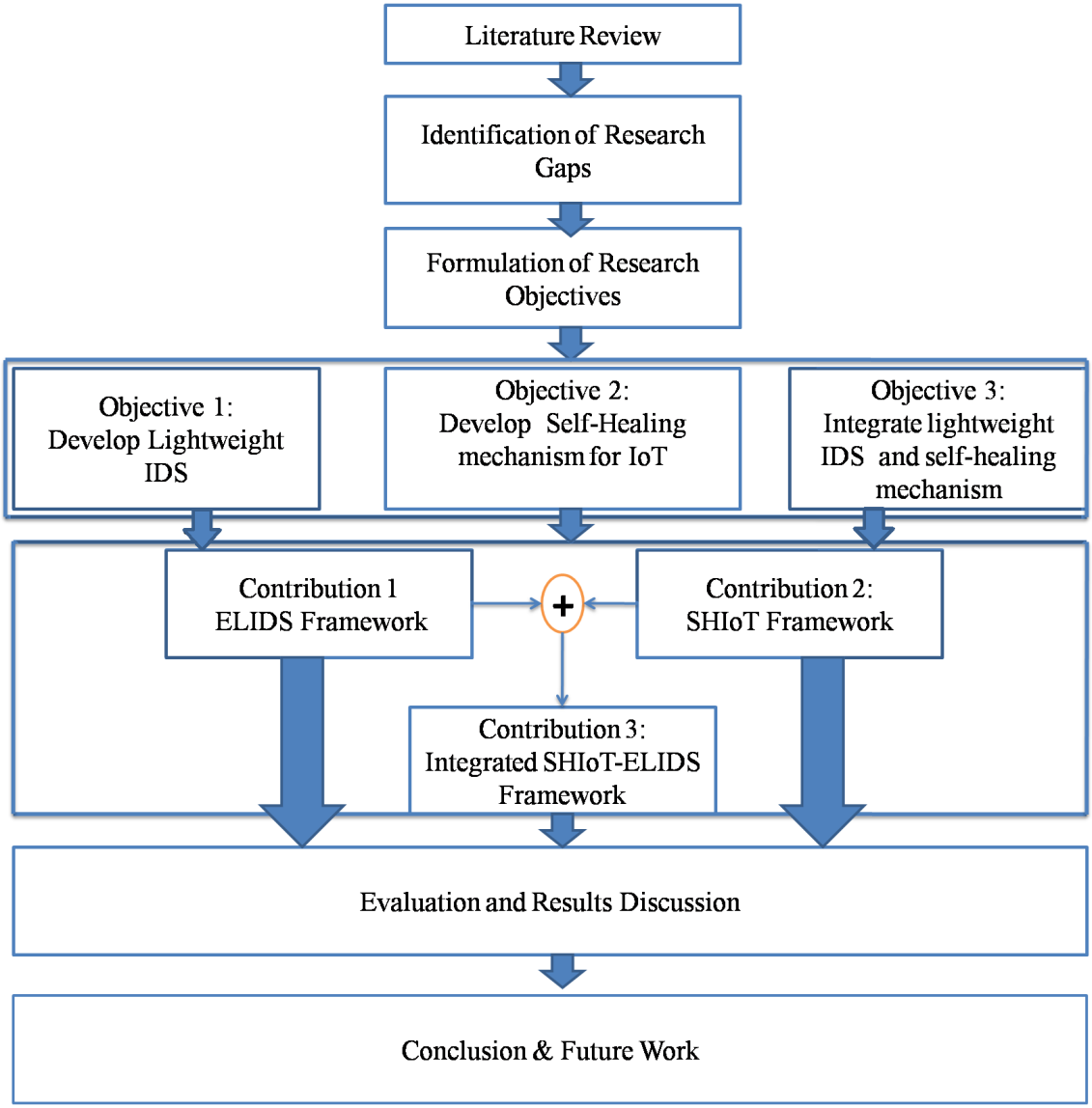


Figure 1.3: Research workflow diagram of the study.

Chapter 4 The outcome of the proposed ensemble FS approach is presented, involving both in-domain and cross-domain performance evaluation. The evaluation

of the approach in two domains aims to ensure the efficacy of the approach in practice.

Chapter 5 Discussion continues to examine another crucial basic element in the proposed framework: the concept of self-healing. The chapter illustrates the specifications and operational aspects of the concept of self-healing.

Chapter 6 Contains comprehensive results relevant to the self-healing process as confirmed by simulated experiments within the realm of the IoT. The study provides a list of results and a thorough explanation for the self-healing process, which is detailed to provide key results to fully comprehend the usefulness of self-healing processes.

Chapter 7 integrates a lightweight and self-healing mechanism to propose the SHIoT-ELIDS (**S**elf-**H**ealing for **I**nternet of **T**hings with **E**nsemble based **L**ightweight **I**ntrusion **D**etection **S**ystem). This chapter also presents the evaluation results and a discussion of SHIoT-ELIDS by employing unseen datasets.

Chapter 8 concludes with a discussion of potential future work.

CHAPTER 2

Background and Literature Review

This chapter describes the background and literature review in relation to the thesis work that forms the motivational aspect of research. This chapter discusses various types of cyber-attacks on IoT networks, focusing on Denial of Service (DoS) and Distributed Denial of Service Attacks in more detail in relation to their classifications in the IoT environment. This chapter continues with the description of the IoT and its inherent challenges in terms of processing power, memory, and energy in relation to IoT devices. A taxonomy of IoT has been provided that classifies IoT devices based on their processing capacities, communication attributes, and application fields.

The review then discusses the subject of Intrusion Detection Systems (IDS) and briefly addresses the present trends for IDS in the context of IoT networks. It also addresses the current developments in the detection of DoS/DDoS attacks and

gives a brief introduction to the basic machine learning (ML) methods being used in this area. The chapter also surveys the state-of-the-art datasets being used to test the effectiveness of the lightweight IDS based on ML algorithms in a cyber-attack setting. In continuation to this, it also includes design methodologies of a lightweight IDS and gives a thorough review of the most recent state-of-the-art methods using ML algorithms to detect cyber-attacks in resource-constrained networks in the context of the IoT. The final topic discussed in this thesis involves self-healing in the Internet of Things and its associated types and self-healing IDS; finally ending with a review of open research and future work in this research area.

2.1 Internet of Things

Internet of Things (IoT) refers to the network of connected devices, ranging from sensors and smart devices to wearables and vehicles, with the ability to communicate with each other and share information without human intervention [29]. In an IoT system, communication begins at the Perception Layer, whose primary responsibility is perception and gathering information from the physical world. Devices on this layer sense physical quantities such as temperature, humidity, movement, pressure, and location. A final sensed piece of information is relayed through the Network Layer, whose responsibility is creating communication infrastructure for the transfer of information from the perception layer and above levels within the IoT system [30]. These layers communicate multiple communication standards such as Wi-Fi, Bluetooth, Zigbee, LoraWAN, and 5G networks. Consequently, the information eventually gets to the Application Layer, whose primary duty is higher-level

processing, decision-making, and value addition for the user or an organization. Application Layer comprises software applications, analytical processing units, and analytical cloud units capable of aggregating all the above information with analytical tools capable of generating new information [31]. Together, all three layers make up the basic structure required for an IoT system's successful functionality and automation [31].

2.1.1 Resource Limited IoT Devices

The IoT devices have sensing and communication capabilities, which facilitate applications such as temperature measurement, pressure measurement, motion sensors, and a host of other sensing applications based on the kind of application being implemented. Nevertheless, the constraints of the computational capabilities of the IoT devices can be attributed to the kind of compromise required with respect to factors such as cost, energy utilization, and the specific applications for which these devices have to be designed [32]. As a result, the restrictive capabilities of IoT devices increase their vulnerability to cyber attacks and decrease their effectiveness in detecting, and defending against such attacks.

In order to counter the challenge of resource limitation in the context of IoT, Table 2.1 below illustrates some of the best examples of the different types of end and gateway devices that have been produced by five different suppliers based on their capabilities in terms of resources such as CPU speed, memory size, and power supply. Note that some suppliers have detailed information on the resources, while others only have information on one or two of the requirements.

Table 2.1: Overview of various IoT end and gateway devices

Company	Product	Device Type	Applications	Resources
Milesight	Indoor Ambience Sensor AM100	End Device	Environment	Power: 2 × 2700 mAh Li-SOCl2 replaceable batteries (7–9 Hrs)
	UG63 Mini LoRaWAN Gateway	Gateway		Power: 12000mAh high-capacity battery (up to 32 hours) CPU: ARM Cortex-A7 (528 MHz) Memory: DDR4 RAM (256 MB) + eMMC Flash (4 GB)
AMobile Sol. Corp.	IB003 Smart Water Meter	End Device	Water Supply	Power: AC + Internal Backup Battery (5 days) MCU/CPU: 32-Bit ARM Cortex (480 MHz)
	G350 Gateway	Gateway		Power: DC (12V) CPU: MediaTek 4X Arm Cortex-A53 (2.0 GHz) Memory: RAM (4GB) + Flash (64GB)
Four-Faith	F8914 ZigBee Terminal	End Device	Manufacturing	CPU: Industrial ZigBee Processor
	F-G100 Intelligent Gateway	Gateway		Power: DC (9–36V) CPU: Industrial-grade 32-bit Communication Processor Memory: DDR3 (512MB) + Flash (32MB)
Zoll	Heart Failure Management System	End Device	Healthcare	Power: Rechargeable Battery
	Smartphone-sized Gateway Device	Gateway		Power: Rechargeable Battery (up to 18 Hrs)
Biz4intellia	Intellia Light Sensor INT-Light-02	End Device	Agricultural	Power: 19000mAh Li-SOCl2 battery (up to 5 Years)
	Intellia Indoor LoRaWAN Gateway	Gateway		Power: DC (9–48V) CPU: 64-bit ARM Cortex-A53 (800 MHz) Memory: DDR3 RAM (512 MB) + Flash eMMC (8 GB)

Generally, IoT end devices have less resource capacity than gateway devices. The devices listed in Table 2.1 have restricted capacity in one or two resources and, therefore, are not suitable for the implementation of resource-excessive security solutions such as conventional IDS/IPSec solutions. This hinders the secure operation of devices and increases the possibility of attacks by attackers who can easily take advantage of such vulnerabilities to perform critical cyber attacks. Additionally, a significant number of devices have network communication technologies with restricted bandwidth, which can easily be attacked using network saturation attacks such as those using DDoS attacks. However, some devices in the IoT domain have enough resources to enable them to implement strong security hardware and

software solutions. These devices include G350 Gateway devices manufactured by AMobile Solution Corp. However, their implementation might be restricted.

2.1.2 Attacks in IoT based Systems

The rising use of IoT networks is vulnerable to various cyber-attack types because of their extensive usage and lack of proper security frameworks. These types of cyber-attacks target various layers within the overall architecture that comprises IoT networks. For the perception layer, the significant risks include those concerning spoofing, tampering, calibration attacks, sensor nodes, and replay attacks. In relation to the network layer, the most popular cyber-attacks include Man-in-the-Middle (MITM), Denial of Service/Distributed Denial of Service (DoS/DDoS), Eavesdropping, Traffic Analysis, and Jamming [33]. In respect to the application layer, risks include SQL injection, Cross-Site Scripting, Data injection, and more, encompassing possible risks to manipulate unrestricted system access, data breaches, and system manipulation [34]. The overall scheme depicting various types of cyber-attacks within a broader classification encompassing five different parameters is shown within Figure 2.1.

Based on Target: Device-level attack encompasses activities like credential theft, firmware tampering, as well as circuit tampering. Network-level attack includes activities like traffic interception, network injection, as well as other network disruptions, which may encompass activities like data leakage, identity theft, and so on. Data-level attack, on the other hand, includes activities like data breach, data damage, with privacy being the main focus.

Based on Attack Vector: Attacks are categorized into application layer, network layer, and physical layer attacks [Zaman, 2009]. Application-layer attacks comprise Command Injection Attacks, SQL Injection Attacks, and Cross-site Scripting Attacks. Network-layer attacks comprise Denial of Service Attacks, Man-in-the-Middle Attacks, Spoofing Attacks, and Eavesdropping Attacks. Physical-layer attacks comprise Hardware Tampering Attacks and Side-channel Attacks.

Based on Attack Methodology: The Figure 2.1 identifies active and passive attacks. Active attacks include injection and replay attacks that actively interact with the network. Passive attacks include traffic analysis that involves monitoring information without changing what is happening on the system.

Based on Impact: The impact of cyberattacks can be categorized into operational impacts and financial impacts. Operational impacts usually involve disruption and lack of functionality, and can lead to system failure or absence of critical services. But financial impacts involve direct economic losses and reputational damages.

Based on Threat Source: Attacks can also be categorized into external attacks, usually by hackers and cyber-criminals in pursuit of personal agendas, and internal attacks, due to staff members in an organization, usually disgruntled employees seeking to use known vulnerabilities in order to steal or damage data.

The need to address these challenges is made more paramount by the increasing convergence of IoT devices with critical and daily life infrastructure. With the rising adoption of IoT technology, there is a critical need for the development and implementation of measures that can protect these devices from threats in their

increasing numbers. Such measures must adopt a holistic approach in providing mitigative and detection mechanisms particularly for IoT.

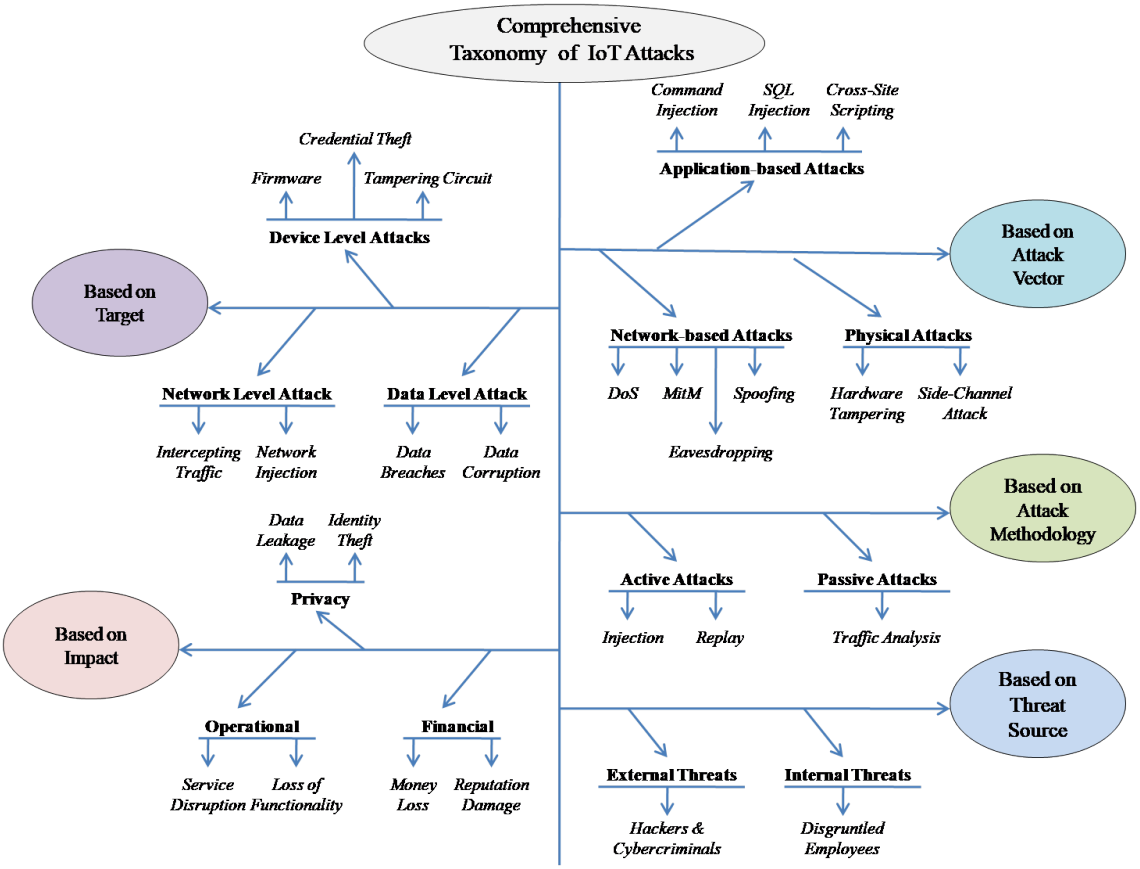


Figure 2.1: Attack taxonomy in IoT

2.2 DoS/DDoS Attacks

During a DoS attack, the attacker floods the victim with a large amount of Internet traffic, making the system unavailable to its users. DoS attacks originate from flooding the victim with traffic from only one source, while DDoS attacks use several compromised hosts, called bots, to flood the victim concurrently. DDoS attacks have become common, even targeting multinational corporations of colossal size. An example of this threat was the biggest recorded DDoS that specifically attacked Amazon Web Services (AWS) in February 2020. The impacts of DDoS attacks

include enormous reductions in legitimate traffic, loss of business opportunities, and reputation damage [35],[36],[37]. The distributed nature of DDoS attacks offers several strategic advantages to the attacker:

- **Increased Disruptive Potential:** Because more machines are being utilized, the adversary can perform more disruptive attacks, thus overwhelming the target's resources and ultimately resulting in a significant level of service disruption or denial.
- **Anonymity and Complexity:** The random scatter of attacking hosts may range globally and could include otherwise legal hosts as it becomes difficult to trace the attack source.
- **Difficulty in Mitigation:** When there are several attacking machines that need to be mitigated and held back from causing harm to a computer system or network.
- **Obfuscation of the Attacker Identity:** This refers to concealing attackers' identities behind multiple compromised systems, making it difficult to trace the root source of an attack.

2.2.1 Types of DoS/DDoS Attacks

The DoS/DDoS are typically divided into three major types: Volumetric attack, Protocol attack, and Application layer attack[38].

- **Volumetric Attack:** A volumetric disruption, also collectively known as DDoS, is the most common type of this kind of cyber threat. The first vol-

umetric disruption to gain widespread interest was the 1999 Trinoo DDoS attack on the University of Minnesota [39], then large-scale attacks in the start of 2000 on well-known online services such as Yahoo, CNN, and Amazon [40]. These are often considered "floods" that flood "a victim server with an incredible amount of data, including unwanted pings." Volumetric Disruptions are measured in bits per second (bps) or Gigabits per second (Gbps) that have been known to go into terabits per second [41], greatly crippling even well-fail-safe systems. These volumetric disturbances often make use of amplification attacks to achieve this. Domain Name System (DNS) amplification attacks involve the attacker sending small amounts of data in the victim's IP address to a DNS server, then the server sending large amounts of data to the victim, flooding their network [41]. It also makes use of both UDP and ICMP flood attacks that make use of the connectionless property of UDP to flood [41].

- **Protocol Attack** Attackers use the inherent flaws in network protocols to launch service attacks, especially focused on particular TCP/IP layers because of their crucial functions in network communications. The Transport Layer (Layer 4) of the OSI model, Transmission Control Protocol (TCP), is attacked by SYN flood attacks. An attacker sends a flood of SYN packets to a server to initiate multiple connection requests but fails to complete the TCP three-way handshake handshake mechanism. Every SYN packet consumed server resources because the host reserved memory to handle the connection request and sent a SYN-ACK in return. The handshake mechanism was not completed, and thereby the resources were held, gradually consuming the server capacity

to serve legitimate connection requests, resulting in a denial-of-service state [37].

Also, Domain Name System (DNS) amplification attacks take advantage of the vulnerabilities in the DNS system, which operates at the Application Layer but uses the Transport Layer to communicate. The attacker then sends DNS inquiries to an open and vulnerable public DNS resolver using a forged source IP address—the victim's IP address. The vulnerable public DNS server then provides an unusually large response to the victim, resulting in an amplification of network traffic sourced at the victim [36]. Additionally, Smurf attacks operate at the Network Layer and make use of the Internet Control Message Protocol (ICMP). For this kind of attack to occur, the attacker broadcasts ICMP Echo Request (Ping) messages to the network's broadcast address using a forged source IP address—the victim's IP address. Consequently, all machines in the network reply to the victim with ICMP Echo Responses, resulting in an enormous response flood that congests the victim's network due to an excessive amount of traffic.

- **Application Layer Attack** Application-layer attacks, classified as Layer 7 (L7) Distributed Denial-of-Service attacks in the OSI model, are primarily focused at the Application Layer. This layer is actually responsible for generating and delivering content in response to Hypertext Transfer Protocol (HTTP) requests. The aim of these types of attacks is to exhaust or flood the targeted server utilizing web services. An HTTP flood can be considered a typical example of a kind of application-layer attack. Here, a large amount of HTTP

request messages needs to be sent to the target server; at times, it involves mimicking normal user activities. In fact, it can be likened to a large number of computers constantly refreshing a webpage simultaneously. Though the requests seem to be legitimate at the outset, a large amount affects the efficiency of handling them by the server [36].

2.2.2 Impact of DDoS on Computing Resources

In most cases, a DDoS attack depletes the resources of the attacked entity because it floods them with malicious traffic. The vulnerability to the depletion caused by a DDoS attack is especially high among IoT devices because they have natural limitations when it comes to their processing capabilities, including memory. The limitations associated with these devices make them specially susceptible to being depleted during a DDoS attack.

Table 2.2 presents a synthesized view on the effects of different forms of DDoS attacks on computer resources like CPU, memory, and bandwidth. The table also mentions the communications protocols that are exploited by different forms of attacks. A thorough understanding of this is necessary in designing an effective method to counter the effects of DDoS attacks.

Additionally, the table above shows how specific DDoS attack types put different amounts of pressure on three main system resource categories: CPU, memory, and bandwidth. Specifically, SYN Flood attacks are described as having a high impact on CPU usage and bandwidth, with low affects on memory. Such is consistent with the nature of SYN Flood attacks, as they seek to consume the processing ca-

pacity of the victim server by generating high volumes of TCP connection requests over the network [42]. Other forms of DDoS attacks, such as HTTP Flood attacks, will put roughly equal amounts of pressure on system CPU, memory, and bandwidth resources, consistent with their complex operation at multiple levels on the protocol stacks [43].

The table below, Table 2.2, indicates the normal volume and time for these types of attacks, showing the level at which a DDoS attack may occur. For example, a Domain Name System (DNS) and Network Time Protocol (NTP) attack may be able to create data traffic levels that reach from several hundreds Gbps to Tbps, making it one of the most dangerous types of a DDoS attack [44]. The level at which these types of attacks may be able to continue from minutes to hours contributes to their dangerous level. In particular, it may be seen that although a Slowloris attack may be considered to have a relatively mild level of damage to bandwidth, it may be able to continue from hours to days [45].

Table 2.2: DDoS attacks and their impact on computing resources

Attack	CPU	Memory	Bandwidth	Protocol	Tools	Attack Volume	Duration
SYN Flood	High	Low	High	TCP	LOIC, Hping3, Metasploit	100s of Gbps	Min to Hours
UDP Flood	Low	Low	High	UDP	UDP Unicorn, LOIC, Metasploit	100s of Gbps	Min to Hours
HTTP Flood	Medium	Medium	High	HTTP	LOIC, HOIC, GoldenEye	10s to 100s of Gbps	Min to Hours
ICMP Flood	Low	Low	High	ICMP	Ping, LOIC, Metasploit	10s to 100s of Gbps	Min to Hours
DNS Amplification	Low	Low	Very High	UDP	Metasploit, LOIC, DNS scripts	100s of Gbps to Tbps	Min to Hours
NTP Amplification	Low	Low	Very High	UDP	Metasploit, LOIC, NTP scripts	100s of Gbps to Tbps	Min to Hours
Smurf Attack	Low	Low	High	ICMP	Smurf, LOIC	10s to 100s of Gbps	Min to Hours
Slowloris	Low	High	Low	HTTP	Slowloris, LOIC	-	Hours to Days
Reflected Amplification	Low	Low	Very High	Various	Metasploit, LOIC, Reflection script	100s of Gbps to Tbps	Min to Hours

With regard to the Internet of Things (IoT) environment where a regular characteristic of IoT appliances is low processing power and bandwidth capabilities, the relevance of specially designed defense mechanisms is highlighted by Table 2.2

itself. CPU-intensive attacks like SYN flood attacks and UDP attacks that require a huge amount of bandwidth are a major concern for IoT appliances that could be strained or disabled by such attacks. Also a major concern could be the type of amplification attack that relies on a relatively low request volume but a huge response volume.

2.2.3 DoS/DDoS Attacks in IoT Systems

The rise in the number of interconnected devices has increased the attack surface area; as a result, networks today are more vulnerable to DoS/DDoS attacks. Notably, the increased threat posed by DoS/DDoS attacks is a matter of concern because these networks provide crucial services for a wide range of purposes such as smart homes and industrial automation processes through IoT networks. The following lists a few DoS/DDoS attacks spotted on the IoT networks recently.

- In a more sophisticated attack initiated in June 2016, over 25,000 DVRs and CCTV cameras were hacked, targeting the website of Bricks Mortar Jewelry Stores. It consisted of about 50,000 HTTP requests per second. Since most websites only handle a few hundred to a few thousand simultaneous connections at a time, the attack made the website unusable. Also, the botnet was found to be spread all over the globe. It predominantly consisted of Taiwan (24
- In September 2016, a large DDoS attack occurred on KrebsOnSecurity, a site run by security consultant Brian Krebs, which reached as high as 665 Gbps. The attack traffic employed SYN floods, ACK floods, GET/POST floods,

and Generic Routing Encapsulation (GRE) floods; these are volumetric DDoS attacks that flood the targeted site's network bandwidth and resources with a huge number of GRE packets. The botnet network employed for the attack was composed of compromised IoT devices such as IP cameras, thermostats, routers, and light bulbs [46].

- In April 2017, a new botnet named Persirai IoT was discovered, with similar codes to the Mirai botnet. It affected over 1,000 IP camera models, with a total of 122,069 affected devices globally. The attackers used a vulnerability in TCP port 81 and Universal Plug and Play (UPnP) to inject commands and download malicious scripts in memory using command injection techniques. Later, it self-deleted and spread using a "zero-day vulnerability" to grab passwords and login credentials.

- GitHub suffered a Distributed Denial of Service attack of 1.3 TBps, which equals 126.9 million packets/sec in February 2018, a DDoS attack. Despite the use of DDoS protection, the event was detected in ten minutes [47].

- One of its clients in the entertainment industry in 2019 had been targeted by a DDoS attack launched through a botnet with more than 400,000 IoT devices in Brazil [48]. For 13 days, these devices made 292,000 requests per minute. The attackers made use of the organization's agent in obtaining access to the authentication module. The attack had continued despite ML defense mechanisms such as CounterBreach 2.0 [34].

- Amazon Web Services was hit by a Distributed Denial of Service attack of about 2.3 Tbps, which used methods such as amplification, reflection, and

botnets [49].

- In 2021, Microsoft Azure witnessed a DDoS attack of around 3.47Tbps in 2021, which is one of the biggest DDoS attacks launched against a cloud service provider; it was conducted using botnets and amplification techniques [50].
- In October 2024, Cloudflare faced a "record-breaking" DDoS attack that reached 5.6 Tbps, launched from a Mirai botnet consisting of over 13,000 compromised IoT devices and utilizing DNS amplification methods [51].

The recent DDoS attack exemplifies the rising trend in the number of DDoS attacks targeting the IoT network. Such attacks involve various approaches to overwhelm the network services using the inherent characteristics of the vulnerable IoT devices. This further emphasizes the need to design an efficient security mechanism to counter the DDoS attack.

2.3 Intrusion Detection System

Based on this, an IDS is a cutting-edge security solution with functionality in monitoring and analyzing network traffic or system events for signs of unauthorized access, malicious activity, or possible security breaches [38]. The major purpose of using an IDS is identifying and warning about possible intrusion attempts that may threaten the integrity, confidentiality, or availability of the protected information systems [38]. By monitoring and consistently analyzing data flows and system logs, an IDS plays a critical part in maintaining a security status in an organization and successfully countering possible security threats at protected premises [38].

2.3.1 Types of Intrusion Detection System

IDS are basic building blocks of network security and can be classified in a number of ways. Such classifications help in understanding the working strategies of IDS systems in an efficient manner.

The major categories involved in the classification of IDS include detection techniques, architectures, methods of detection, and models of deployment [52]. These categories emphasize the various techniques and tactics designed for intrusion detection based on the security needs in the network environment. The different types of IDS and their descriptions are given in Table 2.3. The combination of these types is vital in creating network security.

Table 2.3: Types of Intrusion Detection Systems

Type	Detection Methods	**Signature-based**: Based upon known pattern and sequence of events. Detects malicious activities for both host and network using predefined patterns, similar to antivirus software.
		Anomaly-based: Detects new or zero-day attacks by differentiating normal and abnormal activities using a baseline. Capable of detecting unknown attacks by observing deviations from normal behavior.
	Deployment Architecture	**Centralized IDS**: Placed centrally in the network, managing data and control from a single point. A central IDS monitors and manages network threats.
		Distributed IDS: Deployed across different network locations, working collectively to detect threats. Provides scalable coverage.
	Detection Approach	**Passive**: Monitors and logs network traffic and system activities without intervening. Acts as a silent guard, logging and analyzing system activities.
		Active: Monitors, detects, and takes actions in response to potential threats, such as blocking traffic or alerting administrators.
	Deployment Methods	**Network-based**: Monitors traffic at the network gateway, generating alerts for suspicious activities. Captures network packets to detect malicious patterns across IP headers.
		Host-based: Monitors activities within a host, generating alerts for unusual actions. Dedicated to securing individual hosts, detecting insider threats.
		Hybrid: Combines Network and Host-based IDS features for a more extensive security coverage. Offers characteristics of both approaches.

2.3.2 IDS in IoT based Systems

Intrusion Detection System are very essential in safeguarding the networks created in connection with the Internet of Things (IoT) in relation to detecting and responding to intrusions that constitute violations of security policies in place in the network. The primary purpose of an IDS system is to improve the level of network security by constantly analyzing data flows in order to detect security anomalies.

IDS operate in a passive manner, where they inspect the network traffic by receiving packets of information as they travel within the network environment. They analyze these packets to identify patterns or behaviors that deviate from the set baseline, thus providing an indication of vulnerabilities or breach of security [53]. The IDS system is capable of detecting many threats, most of which have the ability to affect either the confidentiality, integrity, or availability of the network environment.

2.4 Machine Learning

ML can be defined as a subset of artificial intelligence (AI), and it involves building models that can be used for learning from data and making decisions based on said data [54]. As opposed to traditional computer programming where computers are explicitly programmed for particular tasks, ML involves teaching systems massive amounts of data for the purpose of learning patterns and prediction [9]. There are three major classes of ML, and these include [9]: (i) Supervised learning, (ii) Unsupervised learning (iii) Reinforcement learning. These methods involve the use

of statistical techniques for building models capable of learning and generalizing from the data they have been trained with to new unseen data [55].

MH is ubiquitously used in various applications such as image and speech recognition, language processing, and predictive models. Additionally, its ability to analyze large amounts of information effectively and identify complex patterns has made it an essential technology with broad applications across many disciplines. A specific area that reflects the significance achieved by MH is cybersecurity, specifically intrusion detection system [52].

With the emergence of rapid growth in cyberattacks, conventional security systems, which rely solely upon predetermined rules and signature-based detection, tend to be ineffective in recognizing sophisticated threats. This is because conventional security systems are not able to cope with increasing patterns and tend to be less efficient against zero-day threats. But in contrast, ML-based IDSs use sophisticated algorithms that analyze network traffic and system activity in real time.

By leveraging the possibilities of supervised learning, unsupervised learning, and deep learning methods, ML-based IDS can detect threats with higher accuracy and adaptability. These networks possess the capability of handling a large amount of data effectively and actively improving their defenses against potential threats. In view of the evolving nature of cyber threats, the need for incorporation of ML-based solutions within IDS has remained imperative for reinforcing the framework of cyber security [56].

2.4.1 Supervised Learning

Supervised machine learning techniques require the use of labeled examples where each example is accompanied by a predetermined label. During the learning phase, the ML algorithm connects the features of the examples to the target output labels. This increases the ability of the model to predict the classes of unseen examples rather accurately.

In intrusion detection situations, supervised learning is set to play an extremely important part. These kinds of technologies identify known threats by matching traffic with pre-defined signatures of known attacks stored in a database. As new kinds of threats emerge at constant intervals, supervised learning techniques always need updation with new signatures to continue their efficiency [57]. It includes multiple algorithm groups, with descriptions provided below in Table 2.4.

Table 2.4: Supervised learning algorithms

Algorithm	Advantages	Disadvantages
Linear Regression [58]	- Simple and interpretable. - Efficient to compute. - Works well with linear relationships.	- Assumes linearity between features and target. - Sensitive to outliers. - May underfit data.
Support Vector Machines [55]	- Effective in high-dimensional spaces. - Robust to overfitting. - Can handle non-linear data with kernel trick.	- Memory and computation intensive. - Difficult to interpret. - Requires careful parameter tuning.
Decision Trees [59]	- Easy to interpret and visualize. - Can handle both numerical and categorical data. - No need for feature scaling.	- Prone to overfitting. - Can be biased towards features with more levels. - Unstable with small data changes.
Random Forest [60]	- Reduces overfitting. - Handles missing values. - Provides feature importance.	- Less interpretable. - Can be computationally expensive. - Slower to train than single decision trees.
Gradient Boosting [61]	- Often provides high predictive accuracy. - Can handle different types of data. - Handles complex data patterns.	- Computationally intensive. - Requires careful tuning of parameters. - Can be prone to overfitting.
AdaBoost [62]	- Focuses on improving weak classifiers. - Can improve the performance of simple models. - Often results in high accuracy.	- Sensitive to noisy data. - Can be prone to overfitting. - Requires careful tuning.
k-Nearest Neighbors [30]	- Simple and intuitive. - No training phase. - Effective with small to moderate-sized datasets.	- Computationally expensive during prediction. - Sensitive to the choice of k and feature scaling. - Struggles with high-dimensional data.
Naive Bayes [63]	- Simple and fast. - Works well with small datasets. - Performs well with categorical features.	- Assumes independence between features (naivety). - May not perform well with correlated features.
Feedforward Neural Networks [64]	- Capable of learning complex patterns. - Flexible with various architectures. - Suitable for large datasets. - Prone to overfitting if not properly regularized.	- Requires large amounts of data. - Computationally intensive. - Difficult to interpret.
Convolutional Neural Networks [65]	- Excellent for image and spatial data. - Capable of automatic feature extraction. - Handles large datasets well.	- Requires substantial computational resources. - Complex to design and tune. - Large model sizes.
Recurrent Neural Networks [57]	- Effective for sequential data. - Capable of capturing temporal dependencies. - Useful for time series and language processing.	- Prone to vanishing and exploding gradients. - Computationally expensive. - Difficult to train.

2.4.2 Unsupervised learning

Unlike supervised learning techniques, unsupervised learning works on data without any labeled examples. Instead of discovering patterns based on examples provided, unsupervised learning algorithms aim to discover patterns based on inherent characteristics of patterns. It can be highly beneficial for anomaly-based IDS systems,

which require the detection of new threats. By analyzing data points that do not belong to normal patterns, researchers can notice new patterns of attacks that are not programmed in the IDS system. [57] A list of some popular unsupervised learning algorithms along with their benefits and drawbacks is given in the table below (Table 2.5).

Table 2.5: Unsupervised learning algorithms

Algorithm	Advantages	Disadvantages
K-Means Clustering [66]	- Simple and easy to implement.	- Requires specifying the number of clusters.
	- Efficient with large datasets.	- Sensitive to initial cluster centers.
	- Works well with spherical clusters.	- Struggles with non-spherical shapes.
Hierarchical Clustering [67]	- Provides hierarchical relationships.	- No need to predefine cluster count.
	- Can handle different types of distance metrics.	- Computationally expensive for large datasets.
	- Useful for small to medium-sized datasets.	- Sensitive to noise and outliers.
DBSCAN [68]	- Can find clusters of arbitrary shape.	- Needs two parameters: epsilon & min points.
	- Robust to noise and outliers.	- Performance declines with high-dimensional data.
	- No need to specify cluster count.	- May struggle with varying density clusters.
Gaussian Mixture Models [69]	- Provides probabilistic clustering.	- Assumes data follows a Gaussian distribution.
	- Can model elliptical clusters.	- Requires specifying the number of components.
	- Flexible in terms of cluster shape.	- Sensitive to initial conditions.
Principal Component Analysis [70]	- Reduces dimensionality while retaining variance.	- May lose interpretability of features.
	- Useful for visualization and noise reduction.	- Assumes linear relationships between features.
	- Computationally efficient for large datasets.	- Not suitable for non-linear data patterns.
t-Stochastic Neighbor Embedding [71]	- Excellent for visualizing high-dimensional data.	- Computationally expensive for large datasets.
	- Preserves local structure well.	- Results can vary between runs.
	- Can handle non-linear relationships.	- Requires careful tuning of parameters.
Autoencoders [72]	- Can learn non-linear dimensionality reduction.	- Requires significant training data.
	- Flexible architecture for various types of data.	- Computationally intensive to train.
	- Can be used for feature extraction.	- Prone to overfitting if not properly regularized.

2.4.3 Reinforcement learning

Unlike supervised and unsupervised learning methods, Reinforcement Learning (RL) is associated with an agent interacting with its environment whereby it learns from the results associated with its acts. This is achieved through feedback from an environment in the form of rewards or penalties received and fed back to the agent with the aim of maximizing cumulative rewards [54]. Regarding intrusion detection system and their defense against threats, RL can be used for adaptive responses

against ever-changing threat environments. For example, an IDS designed using RL can be acquiring and improving its detection policies from feedback received from the ever-changing threat environments, making it efficient in detecting and protecting against threats over time. This adaptability feature finds RL most appropriate in complex environments where threats keep changing from time to time [52]. RL uses algorithms, which have been enumerated in Table 2.6, including their respective strengths and weaknesses.

Table 2.6: Reinforcement learning algorithms

Algorithm	Advantages	Disadvantages
Q-Learning [73]	- Simple and easy to implement.	- Needs state and action space discretization.
	- Off-policy enables learning from simulations.	- Can be slow to converge in large state spaces.
	- Works well for small to medium-sized problems.	- Struggles with continuous state/action spaces.
State-Action-Reward [74]	- On-policy, which leads to more stable learning.	- Requires careful exploration-exploitation balance.
	- Handles stochastic environments well.	- Can be slower to converge than Q-Learning.
	- Simple to implement.	- Performance is sensitive to the choice of policy.
Deep Q-Networks [75]	- Handles large state spaces with neural networks.	- Needs high computational power for training.
	- Can learn from high-dimensional sensory inputs.	- Prone to instability and divergence in training.
	- Uses experience replay to stabilize learning.	- Requires careful tuning of hyperparameters.
Policy Gradient Methods [76]	- Optimizes policy directly for flexibility.	- High gradient variance requires large data.
	- Suitable for continuous action spaces.	- May get stuck in local optima.
	- Can handle stochastic policies.	- Requires careful tuning of the learning rate.
Actor-Critic Methods [77]	- Merges policy gradient and value-based advantages.	- Can be complex to implement and tune.
	- Reduces variance in gradient estimates.	- Sensitive to architecture and hyperparameters.
	- Suitable for continuous action spaces.	- Can suffer from stability issues during training.
Proximal Policy Optimization [78]	- Balances exploration and exploitation well.	- Can require significant computational resources.
	- More stable and efficient than policy gradients.	- Depends on the choice of clipping parameter.
	- Suitable for large and complex environments.	- Suffer from instability in certain environments.
Trust Region Optimization [79]	- Ensures monotonic improvement in policy performance.	- Computationally expensive.
	- More stable training process than vanilla policy gradient.	- Difficult to implement and tune.
	- Effective in complex environments.	- Less flexible compared to PPO.
Asynchronous Actor-Critic [80]	- Allows for parallel training, improving efficiency.	- Requires significant computational resources.
	- Reduces correlation in updates, improving stability.	- Can be complex to implement.
	- Suitable for large, complex environments.	- Relies on architecture and hyperparameters.

2.4.4 Machine Learning enabled IDS

Intrusion Detection System aided with ML use sophisticated ML techniques to detect and defend against possible security threats in computer networks. However, the increasing Internet of Things (IoT) adoption within different domains has am-

plified the difficulty level of securing these systems [81]. Therefore, it is crucial to implement efficient security systems to allow the transformative power of IoT to be fully harnessed. In this respect, ML-assisted IDS have been identified as a dominant technique to secure the entire IoT environment from malicious users [62]. This is achieved by gathering data examples with both benign and malicious data to train the ML model for IDS.

Many ML-assisted models, including supervised and unsupervised learning techniques, have been designed to effectively detect new attacks within the IoT environment [82], [83]. With supervised learning, the model trains using labeled examples and is able to distinguish between normal and malicious network traffic based on this training. In contrast to supervised or unsupervised machine-learning techniques using labeled or unlabeled data to distinguish between normal and malicious network traffic; Host-based IDSs extend this logic to system-level data like logs and processes. Host-based IDSs are generally either signature-based or anomaly-based systems. In signature-based systems, they hold patterns of malicious activities; in anomaly-based systems, they learn normal network behavior to establish a baseline in order to trigger alarms in instances of deviation [84].

The ML-assisted IDS is most suited for environments where IoT is predominant owing to their adaptability functionality against the omnipresent and ever-changing nature of IoT networks [65]. However, one of the challenges faced by today's IDS is the incidence of high false positives, hindering operations in their wake. Yet, in order to address this problem, new approaches are being explored concerning the use of advanced ML and DL methods [62]. Moreover, its effectiveness in securing

networks is being appreciated owing to their ability to counter ever-evolving adversaries. Securing information assets and networks against these malicious activities is vital. Also, traditional IDS systems can be upgraded with new approaches concerning zero-day attack detection via the integration of ML algorithms, assuming an essential role in detecting new zero-day variants [82].

2.5 Benchmark Datasets for Designing ML enabled IDS

A key role has been played by datasets in building and training classifiers in IDS for the detection and categorization of different types of cyber attacks [85]. Datasets form the basis of the creation of reliable IDS classifiers and are vital to the training and evaluation processes in the detection of different types of cyber threats. A range of publicly available datasets has been made available to the research community in a manner that has contributed to the advancement of IDS technology. These include KDDCup99, NSL-KDD, TON_IoT, CICDDoS2019, WSN-DS, BoT-IoT, and ADFALD datasets. The datasets form the most important elements in the creation of IDS classifiers and in terms of validating the effectiveness and accuracy in the detection of different types of cyber threats [30].

The KDDCup99 dataset, produced by MIT Lincoln Laboratory DARPA, simulates a typical local area network environment of the United States Air Force and is most broadly applied in testing various anomaly detection-based methods in IDS [86]. NSL-KDD dataset is an improved version of KDDCup99 and is applied in coping with redundant and repetitive data that exist in KDD-CUP-1999 [87]. In

this context, UNSW-NB15 is a dataset that models modern network threats and is especially useful in training and testing IDS [88]. In a similar context, dataset CIC-IDS2019 models network traffic data representing various DDoS attacks and is especially useful in testing IDS capabilities in fighting DDoS threats in simulated and actual network settings [89]. WSN-DS is a dataset specifically used in WSN for developing IDS with a focus on network traffic data representing various DDoS attacks and is especially useful in developing IDS in a sensor network setup [90]. Furthermore, ADFA-LD is a dataset particularly useful in modeling modern web-based attacks and is especially important in developing ABH-based IDS with a focus on modern web-based attacks [91]. The most modern dataset related to this context is BoT-IoT, applied in modeling various DDoS threats in Internet of Things (IoT) and is especially useful in developing IDS in an IoT setup with features fully representing actual IoT environment attacks [92]. In a similar context, dataset TON_IoT models actual Industry 4.0/Industrial IoT (IIoT) environment and various attacks in a telemetry setup and is especially useful in developing and testing IDS in a modern IoT setup due to its various formats [93].

This part of the work comprises an in-depth examination of the most widely used datasets, with specific focus on those that have been used in the creation of IDS. Additionally, Table 2.7 encompasses the key aspects of the datasets in terms of the number of data records, number of features, year of publication, number of dataset records in different formats of files, size of the dataset, data gathering techniques, as well as the range of attack classes.

Table 2.7: Comprehensive detail of widely adopted datasets in intrusion detection

Dataset	Records	Features	Types of Attack	Size
KDDCup99 (1999) [86]	4898431	41	Probe, DoS, U2R, R2L	1.5 GB
NSL-KDD (2009) [87]	148517	41	Probe, DoS, U2R, R2L	30 MB
UNSW-NB15 (2015) [88]	2540000	49	Fizzer, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, Worms	100 GB+
CIC-IDS2019 (2019) [89]	1,000,000+	80+	Reflection-based, Exploitation-based	20 GB
WSN-DS (2018) [90]	374661	19	Blackhole, Grayhole, Flooding, Scheduling	60 MB
ADFA-LD (2013) [91]	100,000+	23	Web-based attacks	1 GB
BoT-IoT (2020) [92]	100,000,000+	50+	DoS, DDoS, Port scans, Keylogging, Data exfiltration	500 GB+
TON_IoT (2020) [93]	461,043	45	Scan, DoS, DDoS, Ransomware, Backdoor, injection, XSS, pass-cracking, MiTM	80 GB

2.6 Feature Selection Techniques

Feature selection (FS) methods are widely used to increase the efficiency of the IDS [94]. Such techniques remove the redundant features used in the classification system and help to increase the efficacy of the IDS. FS can be defined as a preprocessing optimization technique used to choose a subset of the most important features and remove any redundant and irrelevant features from a dataset. This allows the building of an efficient learning model and can be used to increase the efficacy of an IDS when implemented in an Internet of Things (IoT) network. It should be noted that a dataset with a large number of features takes a longer time to train and test and consumes plenty of computing resources [95]. However, not all of the features of a dataset are always important for attack detection for a particular IDS classification system. Several methods can be employed to reduce the above-discussed issues, and FS techniques can be used as a pre-processing technique for implementing ML algorithms [96] [97].

There are normally four steps involved in the FS process, as illustrated in Figure 2.2. The steps include the creation of a subset of features based on the original features. This subset is then analyzed using an evaluation function that determines the goodness of the subset. After that, the subset undergoes the stopping criterion decision of whether it should be accepted or rejected. There are three basic types of FS approaches: Wrapper, Filter, and Embedded methods Figure 2.2.

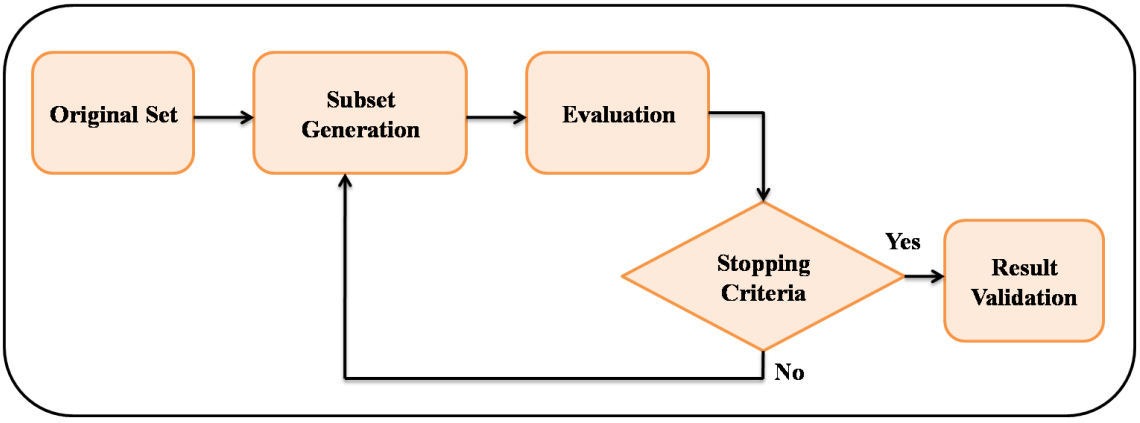


Figure 2.2: Feature selection procedure

2.6.1 Filter Methods

The filter FS technique removes irrelevant features with less effect on data analysis processes. Filter FS comprises conducting a statistical test on each attribute in the dataset, with primary focus on its correlation with the target variable. According to filter conditions, each feature is assigned ratings based on certain statistical tests [97]. Features with the highest ratings are retained, and low-rated features are eliminated. Filter FS is system-resource friendly with low system overhead [98]. Examples of filter FS techniques include Mutual Information (MI), Fisher Score (F-Score), Chi-Square (χ^2), Variance Threshold (VT), Pearson Correlation Coefficient

(R), Mean Absolute Difference (MAD), and Ratio of Dispersion (DR), among others. These will be discussed below:

2.6.1.1 Mutual Information (MI)

This technique measures the degree to which information about a certain attribute improves the prediction of a target variable [99]. MI is generally used in finding the order in which attributes are picked. In other words, MI measures the decrease in the entropy of a target variable when a certain attribute is known. Given a dataset consisting of a target variable y and an attribute x, the formal definition of MI is provided in Equation 2.1.

$$MI(x, y) = H(x) - H(x \mid y) \tag{2.1}$$

where H(x) is the entropy of variable x and H(x|y) is the conditional entropy of attribute x given y.

2.6.1.2 Chi-Square (χ^2))

The χ^2 test is one of the techniques used to determine the association or independence of two categorical variables. The χ^2 test is ideally suited for categorical data with a small number of categories [100]. The chi-squared test primarily establishes the relationship between the two variables based on the occurrence of a given type [100]. The chi-squared formula has been expressed based on Equation (2.2) as below:

$$\chi^2 = \Sigma \frac{(O - E)^2}{E} \tag{2.2}$$

where O represents the observed frequency in a cell and E denoted the expected frequency in the same cell.

The χ^2 test statistically examines whether there is a departure in observed frequencies from expected frequencies and can determine whether a particular association between categorical variables is not random. Such a test has been applied in various areas with the aim of knowing how categorical data relates.

2.6.1.3 Fisher's Score (F-Score)

The F-Score is a statistical measure used to assess the discriminative ability of the features in the data. In a classification problem or pattern recognition problem, Fisher's score helps to determine the most relevant feature to distinguish the different classes or types. The score for a particular feature is computed using the ratio between the class means to the variance within a class, as shown in the equation [101]. The F-Score for a particular data feature X for the two different types, say C1 and C2, can be computed using Equation (2.3).

$$F(X) = \frac{\sigma_1^2 + \sigma_2^2}{(\mu 1 - \mu 2)^2} \tag{2.3}$$

where $\mu 1$ and $\mu 2$ are the means of feature X for classes C1 and C2, respectively. σ_1^2 and σ_2^2 are the variances of feature X within classes C1 and C2, respectively.

2.6.1.4 Pearson's Correlation Coefficient (PCC)

The Pearson correlation coefficient (PCC) represents a statistical index employed for the evaluation of the strength and orientation of a linear association between two continuous variables [102]. It has a range of between -1 and 1, with:

-1 showing perfect positive linear correlation, as the increase in the variables is accompanied by a proportionate rise in the other. - 0 indicating no linear correlation since there is no relationship existing between these two variables.

-1 showing a perfect negative linear relationship between the variables because as one variable rises, its paired variable drops by an equal proportion. The PCC is calculated using the formula analogous to Equation (2.4) above, and the value derived is indicative of both the degree and nature of linear association.

$$PCC = \frac{\sum \left(\left(x_i - \bar{x} \right) \left(y_i - \bar{y} \right) \right)}{\sqrt{\sum \left(x_i - \bar{x} \right)^2 \Sigma \left(y_i - \bar{y} \right)^2}} \tag{2.4}$$

where x_i and y_i are individual data points from the two variables, $\bar{x}$ and $\bar{y}$ are the means of the x and y variables, respectively.

2.6.1.5 Variance Threshold (VT)

A technique in FS where the variance of different features in a dataset is calculated based on a certain threshold [103]. If the variance of the features is less than the threshold, the features are discarded, and features with variance equal to and above the threshold are selected. Mathematically, the above technique can be described

using Equation 2.5. The inclusion of a feature as part of the selection can be controlled using a threshold (T); if the variance of the feature (VT) exceeds the threshold T, the feature can be selected, and if not, it is discarded.

$$VT = \frac{\sum (x_i - \bar{x})^2}{n} \tag{2.5}$$

where x_i represents individual values of feature X. The $\bar{x}$ is the mean of feature X and n is the total number of instances.

2.6.1.6 Mean Absolute Difference (MAD)

The MAD is a statistical measure that calculates a mean absolute deviation between individual data and the center statistic, typically the mean or median, as depicted by [104], as shown by the calculation formula in Equation (2.6) below:

$$\text{MAD } = \frac{\sum_{i=1}^{n} \mid x_i - \text{ central value } \mid}{n} \tag{2.6}$$

where x_i represents individual data points. The *central value* represents the mean or median of the dataset and n shows the total number of data points. The MAD technique calculates the average distance between each data point and the chosen central value. It is often used as an alternative measure to variance or standard deviation, especially when dealing with data that might contain outliers or when a more robust measure of dispersion is desired.

2.6.1.7 Dispersion Ratio (DR)

The DR is another statistical tool employed to determine the measure of variability or dispersion in a set of data, focusing on the spread of the data points with respect to the central point, usually known as the mean or median. The DR measures the spread of the points around the central point, providing guidance on the variability expressed by the set of data around its central point [97]. The DR formula is expressed as in Equation (2.7).

$$DR = \frac{\text{Range}}{\text{Central Tendency}} \tag{2.7}$$

where *Range* represents the difference between maximum and minimum values and the *Central Tendency* is the mean or median, representing the central point around which the data values cluster. The DR has a dimensionless value and a maximum value indicates a larger spread relative to the central tendency whereas, a minimum value indicates a smaller spread.

2.6.2 Wrapper Methods

Wrapper feature selection makes use of the greedy search method that tests all possible feature sets for the chosen evaluation measure [105]. The process analyzes the feature of interest as well as possible feature subsets for the independent variables through the training of the classifier. Even though the accuracy of the method is higher compared to the filter method, the computational cost is higher [97]. Examples of the wrapper method include Forward Feature Selection (FFS), Backward

Feature Elimination (BFE), and Exhaustive Feature Selection (EFS).

2.6.3 Embedded Methods

Embedded techniques merge the strengths of filter and wrapper methods in a way that produces a set of features which are both efficient and very accurate [98] Embedded FS contrasts with filter and wrapper methods in that embedded FS takes place in tandem with training a model. Embedded methods are therefore more efficient when it comes to resource consumption [98].

2.7 State-of-the-art Lightweight IDS

In contrast to general-purpose computers, which typically have substantial storage and computational resources, IoT devices are generally constrained by limited storage and processing power. While IDS solutions can be highly effective in securing traditional IT systems, the resource limitations inherent in IoT devices preclude the use of conventional, resource-intensive IDS approaches. The development of IDS for IoT environments presents significant security challenges due to the constraints imposed by limited memory, processing capabilities, and battery life. Consequently, researchers are focused on designing lightweight and efficient IDS solutions that can effectively secure IoT networks against potential cyber threats while optimizing the use of available resources. According to [106] lightweight systems are primarily designed to conserve energy and minimize computational resource usage. As outlined in [107], a system is considered lightweight if it exhibits reduced energy consumption. Thus, a lightweight system is typically characterized by its ability to perform

necessary computations while using minimal resources. In contrast, [108] defines a lightweight IDS as a compact module that requires only a minimal amount of memory, sufficient to be a permanent component of the Wireless Sensor Network (WSN) security framework. Additionally, [109] describes a lightweight IDS as a security solution that demands minimal computational power, storage, and energy resources while still meeting the required security objectives.

Unlike general-purpose computers that often have large memory and processing capabilities, IoT devices are usually hampered by low memory and processing capacity. Although IDS solutions can prove to be very efficient in protecting traditional computer networks and systems from various cyber threats and attacks, low memory and processing capacity requirements that come with IoT devices make it impossible to apply traditional resource-extensive IDS solutions and techniques. The main challenge that comes with developing an IDS solution for IoT environment is enhanced and/or increased requirements related to memory capacity and battery life of IoT devices. As a result, researchers continue to place high efforts and emphasis on developing efficient and effective lightweight IDS solutions that can play a significant role in protecting IoT networks from various forms of cyber threats and attacks. According to [106], a lightweight system is designed mainly to achieve reduced energy and minimal computational resource requirements. As explained by [107], a lightweight system is expected to demonstrate low energy requirements. Generally, a lightweight system can or should always be defined as a system that performs necessary computations while consuming minimal resources. According to [108], a lightweight IDS can or should always be defined as a small

module that requires minimal memory capacity. This minimal memory capacity is adequate enough to enable a WLAN/WSN safety framework to always remain a constant component and addition. Furthermore, according to [109], a lightweight IDS can or should always be defined as a solution that requires minimal memory and/or energy capacity while still managing to achieve fundamental and principal requirements related to safety objectives.

This section explores various approaches to designing lightweight IDS, balancing efficiency, accuracy, and minimal resource consumption.

2.7.1 Feature Selection based Lightweight IDS

In this section, an intensive analysis of recent lightweight IDS is provided to briefly evaluate their recent developments using FS techniques. Generally, this discussion provides an intensive overview of approaches and outcomes, as well as an intensive analysis of current FS approaches with the aim of comprehending recent intensive developments of lightweight IDS.

[38] worked on an IDS design that has been specifically developed to handle the environment of IoT, in which there is a limitation in terms of computing resources. The proposed system applies a chi-squared filter-based FS technique to identify the most prominent set of features to train ML classifiers such as Random Forest (RF), Decision Tree (DT), Support Vector Machine (SVM), Logistic Regression (LR), and k-Nearest Neighbors (KNN) to classify network traffic as malicious or benign. The evaluation criteria include CPU usage, memory usage, False Positive Rate (FPR), True Positive Rate (TPR), as well as the time taken to execute the algorithms during

training/testing phases. From the results, it has been found that the accuracy of RF was better than all others, not to mention that it used less resources, had lower FPR values, achieved better TPR, and took lesser times than the alternative methods.

Khanday et al. [110] An approach based on FS has been proposed which utilizes the ExtraTreesClassifier in determining and extracting relevant features in a given dataset. The technique uses the RF technique in evaluating feature significance based on Gini Impurity. For BOT-IoT and TON_IoT datasets, the 20 most dominant features were determined, and then features with reduced significance were omitted. The K-best features were then used in building and testing ML classifiers, culminating in the develop- ment of a remarkably efficient intrusion detection system. The research work is mainly focused and centered around building and testing this system for identifying a distributed denial-of- service (DDoS) attack in IoT networks with constrained resources. Nevertheless, there is no mentioned data related to CPU and Memory usage associated with this procedure.

In a recent investigation detailed in [111], a new method using ML algorithms to design a lightweight intrusion detection system (IDS) model is proposed. The research focuses on the use of sophisticated ridge regression approaches to boost feature selection methods. Various approaches, including importance coefficient methods, backward/forward FS, and methods using the correlation coefficient, are employed to identify twelve important features. These twelve features are derived from KDD-CUP-1999, BotIoT-2018, and N-BaIoT-2021 datasets to boost the efficiency and effectiveness of the IDS model. The findings show the efficacy of the proposed system in terms of accuracy, establishing its ability to precisely identify

network intrusion attacks, making it appropriate for resource-constrained devices.

Saheed et al. [112] an ML-based IDS targeting the IoT is presented. This study uses the UNSW-NB15 dataset to design multiple feature-selection (FS) classifiers. As a preliminary dimensionality reduction approach, principal component analysis is used, and 10 out of 44 features are selected for building classifiers. ML algorithms, namely XGBoost, CatBoost, KNN, Support Vector Machine (SVM), Quadratic Discriminant Analysis (QDA), and NB classifiers, are designed and trained using the selected set of features. Results clearly show that almost all classifiers manifest accuracy percentages above 95%. Nonetheless, training times for the CatBoost-based classifier are higher compared with other classifiers. But KNN and SVM classifiers not only produce high accuracy levels, they also introduce smaller training times. Nonetheless, testing times, CPU usage, and memory used in the design of the lightweight IDS presented in this work are not considered.

Roy et al. [113] a B-stacking FS-enabled intrusion detection system (IDS) model was designed for resource-efficient Internet of Things (IoT) networks that leveraged ensemble learning techniques to boost accuracy. In this method, boosting and stacking algorithms were combined effectively for better results. The system used a level-0 learning algorithm that aimed to counter bias before the feature set was transmitted to the level-1 learning algorithm for training the system. Additionally, parallel learning was used for boosting algorithms that aimed to conserve computation cycles and optimize learning rates during training. The datasets NSL-KDD-CUP-1999 and CICIDS2017 were pre-processed with analysis for multicollinearity, sampling, and dimensionality reduction that eliminated unnecessary features, iden-

tified proper training data samples, and prevented overfitting. The results suggested that the proposed system consumed less CPU or RAM during the B-stacking stage of the process with higher accuracy rates and lower false alarm rates compared to existing methods. Although this system was designed for resource-efficient IoT networks, the datasets were not specific to the IoT domain during experimentation and evaluation. In addition, the resource-intensive nature of sampling data, feature reduction, or selection was not included as a factor during resource measurement and analysis.

Sai et al. [114] A lightweight IDS was proposed based on the correlation FS strategy. The method is able to detect the important features that correlate with the target variable. In this study, the UNSW-NB15 dataset was used for training the method. Only three important features were selected among the total of 44. The selected important features were used for training the support vector machine classifier. From the experiment results, the accuracy of the classifier was found to be 98%. However, the study did not measure the resource usage of the classifier performance.

Ozer et al. [115] work comes up with a lightweight cyber attack detection model formulated with four major steps: data gathering, data cleaning, modeling, and evaluation. To accurately evaluate the performance of classifiers, this research work has also been implemented in a real-time setup. To make a system with lesser overhead and maximize accuracy, the authors have cited a novel solution in which unique pairs of features are created using a permutation technique. These unique pairs were then used in identifying a ML algorithm, and from these pairs, the one

with maximum accuracy was considered and analyzed further. The time complexity of these algorithms is calculated in this context. Results proved that Random Forest, Decision Tree, Neural Network, and AdaBoost had maximum accuracy; but these models failed in maintaining their accuracy in a real-time setting. To keep their system lightweight, a total of 12 features were considered, out of which 10 features were selected using a correlation analysis and entropy analysis.

Omar and George [116] A resource-swift method for cyber-attacks in IoT networks was introduced. The method was tested using a laboratory environment comprising indoor and outdoor Wi-Fi security cameras. To promote fast training, conserve memory space, and combat the ill-effects of overfitting in a cyber-attacks detection method, the correlation coefficient FS technique was utilized to eliminate inconsequential inputs. The proposed scheme employed a variety of ML algorithms to classify normal from cyber-attacks traffic, specifically: SVM), DT, RF, Linear Discriminant Analysis (LDA), Stochastic Gradient Descent (SGD), and kNN. Amongst these, the RF performed better in relation to detection accuracy rates. However, none of the algorithms were focused on the model's resource utilisations such as CPU and memory utilized by the model..

Khater et al. [117] An IoT-based light-weight intrusion detection system (IDS) was proposed. The work presents a data exploration phase that involves the examination of the ADFA-LD dataset for distinguishing between the normal and attack packets. During the data preprocessing phase, feature extraction is carried out through the application of the N-gram transformation with the view to include the extracted features for the feature selection phase. Within the proposed framework,

the work utilizes the application of the Mutual Information (MI) method and the Principal Component Analysis (PCA) technique for the purpose of carrying out the FS. The filtered set of features is applied to the training phase for the purpose of training the MLP classifier for the intrusion detection process. During the testing phase, the performance of the proposed MLP classifier is compared with that of the other classifiers for the intrusion detection purpose. These classifiers include the DT, RF, SVM, kNN, and the NB. It is apparent that the RF classifier uses the least amount of memory as compared with the other approaches. Additionally, the RF classifier has the highest accuracy as well as the lowest FPR compared with the other approaches. It is for this reason that the conclusion is drawn that the MLP classifier is not the most suitable choice for the purpose of distinguishing between the classes for the resource-restricted device scenario.

Lee et al. [118] IMPACT (IMPersonation Attack DetecTION) is the optimized classification model designed specifically for the intrusion detection system (IDS). It is composed of three major components: feature extraction, FS, as well as the classification phase. The feature extraction process is performed through the use of the deep neural network-based stacked autoencoder (SAC). Within the FS phase of the system, the model uses the concept of mutual information (MI) as well as the C4.8 wrapper techniques. The feature extraction phase is applied to the 154 original feature inputs of the system. This results in the generation of 50 new feature inputs through the use of the SAC. In the subsequent step of the FS phase of the system, the 204 feature inputs are subjected to the various wrapper techniques. Afterward, the five contemporary characteristic features of the system are fed as inputs for the

SVM classifier in the development of the detection system. Even with the optimized system displaying exceptional results of the classifier for the generic as well as the attack packets, the system is prone to increased rates of false alarms as well as less accurate models compared to other detection models.

Kumar et al. [119] The proposed work describes a misuse-based intrusion detection system (IDS) that is divided into two major parts. The first part relates to the development of an integrated rule-based model of the IDS using the UNSW-NB15 dataset. For the second part, a real-world dataset was created in a virtual setting to measure the efficacy of the developed model of the IDS. For the purpose of improving the accuracy and optimizing the computational complexity, the Information Gain (IG) values were calculated for each attribute, selecting 22 features out of the initial 47 features that all showed IG values greater than zero. A number of decision tree algorithmic variants were combined according to the proposed model, including the C5 algorithm, the CHAID algorithm, the CART algorithm, and the QUEST algorithm. The highest accuracy obtained using the proposed model on the real-world datasets is 83.8 percent. However, the proposed model's potential effectiveness might be impaired because it is a signature-based technique.

Rani and Kaushal [60] Network Intrusion Detection System (NIDS) was also suggested using the Random Forest (RF) classifiers to detect intrusions in the Internet of Things (IoT) network environment. The system was trained and tested using two public datasets: KDD-CUP-1999 and NSL-KDD datasets. Out of the total 41 features, only 10 were chosen to decrease the training and testing time with a resultant accuracy rate of 99.9%. In particular, the authors have not given any

information about the FS techniques used.

The study [120] For IoT networks, an anomaly detection framework is proposed that consists of three phases: (i) Data Aggregation, which aggregates the data produced by the IoT sensors; (ii) Data Preprocessing, which eliminates duplicate entries and uses FS to decrease the dimensionality of the data; and (iii) ML IDS, which uses Support Vector Machines (SVM), Random Forests (RF), Backpropagation Neural Networks (BPNN), and Decision Trees (DT) to detect anomalies in the IoT network traffic. The experiment carried out on various scenarios with self-collected data sets proves that the lightweight IDS achieves promising results in accuracy and training/testing time. The paper fails to analyze the usage of resources by the classification algorithms and ignores factors such as CPU usage, memory usage, and power consumption.

Ahn et al. [121] in this paper, a lightweight intrusion detection system, called Hawkware, is proposed, consisting of two major components: monitor module and detection module. The monitor module is responsible for conducting the analysis of both network and device behaviors to extract key features, while the detection module continually scrutinizes the system for suspicious behaviors, which are characteristic of a network intrusion, utilizing inputs supplied by the monitor module. The intrusion detection mechanism combines both Network Behavior Feature Vectors (NBFVs) and Device Behavior Feature Vectors (DBFVs), with the objective of lowering the processing complexity altogether. NBFVs are responsible for encoding features extracted from the packet headers using a packet analyzer, while the DBFVs encapsulate features linked to system calls. In order to train the models, the pro-

cess was conducted offline, implying a potential existence of biases when the models are operationalized within real-world settings. Additionally, the experiment was conducted using a Raspberry Pi board with ample processing capabilities, thereby raising questions about the suitability of the proposed intrusion detection system for utilization within Internet of Things (IoT) nodes that run within embedded systems with constrained processing capacities.

[122] present a new lightweight intrusion detection system (IDS) designed using fuzzy logic to secure MQTT-IoT networks from DoS attacks in low-configured IoT devices. The designed IDS aims to achieve three primary tasks: (i) DoS attack detection in the IoT networks, (ii) efficiency, and (iii) faster and efficient detection of the DoS attack. The DoS detection mechanism utilizes the fuzzy logic technique to inspect the network traffic through the calculation of the ratio between the Connection Message Ratio (CMR) and the ratio between the Connection Acknowledgment Message Ratio (CAMR). Using these ratios, fuzzy logic rules are applied to identify whether a message is "safe" or "malicious data." The fuzzy logic rules are created using a set of features derived from a "trace dataset," where the critical features are acquired by devices from the network traffic. The paper fails to describe the implementation process performed for the selection of features (FS).

An IDS proposed by [123] is expected to identify DoS attacks, with the use of the J48 classification algorithm. The data set used in the IDS system was collected from practical network environments, where the characteristics of the packets were measured using information gain (IG) to identify the features of primary interest. The IDS model performed well when trained with a distinct setting of 20 features

with a 100

[124] proposed a lightweight attack detection system using a Support Vector Machine (SVM) classifier capable of distinguishing malicious nodes that sought to transmit anomalous data through IoT network traffic. The attack detection system specifically focused only on one feature related to packet arrival rate per nodes in making a detection decision, thus minimizing both time and complexity. Nevertheless, it has proved useful with distinguished capabilities in making a DDoS attack classification using an SVM classifier. The system faced challenges in coping with concurrent actions due to a single feature. Soe et al. (2020) introduced a lightweight anomaly detection system designed for IoT networks. The proposed method uses a correlation feature selection method to identify the top seven features from a total of 49 features. The features were used to train the classification system based on the J48 algorithm. The test process involved identifying the top seven features from the test data. The study presented an average detection rate of 99% based on all attack types.

2.7.2 Ensemble Feature Selection based Lightweight IDS

The traditional methods available for FS, including wrapper, filter, and embedded models [125], address the dimensionality issues effectively [**?**]. This was seen clearly within the context of network traffic analysis, where FS removed irrelevant, redundant, and less informative data components [97]. However, despite the various advantages associated with FS methods, the following limitations must be considered: Biases of the individual might affect FS because different techniques have

different standards of evaluation. Thus, the character sets might vary significantly among different techniques. Using a wrapper model for FS might include a robust model to classify and has been proven to work effectively in some datasets and tasks [99]. Overrelying upon a single technique might cause a sub-optimal subset of the character set because of its vulnerability to local optima problems; this problem becomes worse in terms of multi-objective fitness functions.

However, to rectify these above-mentioned weaknesses, there is great value added through ensemble FS methods. "Ensemble" refers to a group or set of people or elements working or cooperating collectively for a common goal [126]. This will support the saying “two heads are better than one.” Using an "ensemble FS" paradigm for FS (FS) involves multiple approaches like “method_1,” “method_2,” and “method_n,” shown in Figure 2.3 from “[96]” Each will produce its own ranking or collection of feature subsets based upon certain criteria. These can then be consolidated using voting, ranking, or score-averaging methods to provide an overall feature list.

This approach, often used in classification tasks, combines the results of a number of basic classifiers to improve generalization and robustness. While being efficient can be context-dependent, there is evidence that a set of comparatively simple classifiers can rival or outperform a complex one at a lower computational expense than complex models [127].

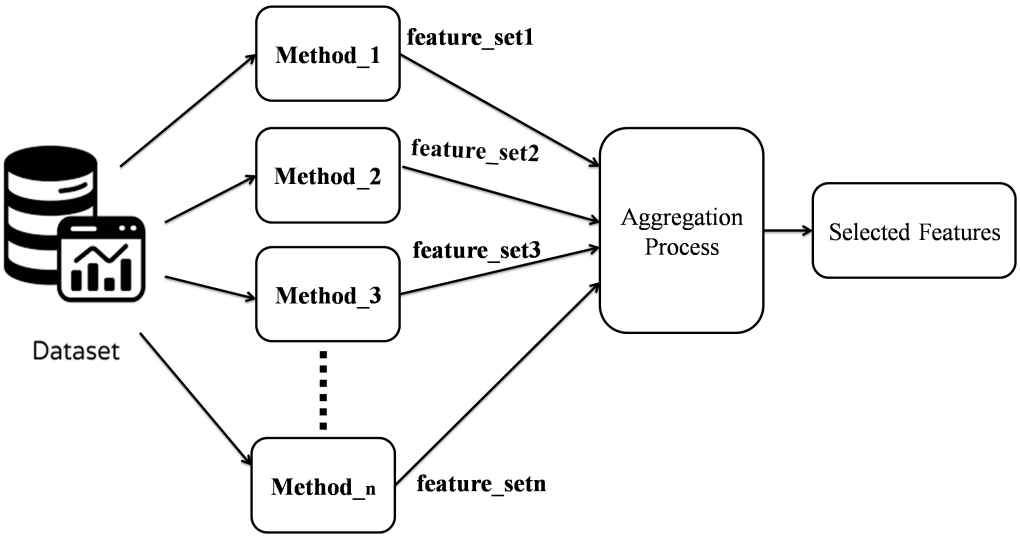


Figure 2.3: Ensemble feature selection

One of the developments that have recently emerged in trying to enhance the efficacy of IDS in Internet of Things devices is ensemble FS. Like conventional FS techniques, the ensemble FS approach offers a reliable mechanism for filtering unnecessary network traffic, which is imperative in light of the inherent limitations of devices in the Internet of Things ecosystem, such as bandwidth capacities.

The concept of ensembles has been extended to various areas of ML, including FS. Underlying all of these approaches of feature set in ensembles is the overarching theme of combining the feature sets of various approaches to leverage their strength. The essential diversity that exists between individual feature set approaches results in a superior feature set that is then further integrated. Traditionally, various approaches as well as various strategies have been devised in the realm of ensembles of FS. Most of the studies in this direction focus on optimal feature set fusion of individual feature set approaches or selecting individual feature set approaches to be considered in the ensemble model.

In [128], A new hybrid method has been introduced for evaluation and fusion of different feature sets chosen through different filter-based FS methods. The newly introduced innovative method, known as "Ensemble Automatic FS (EAFS)" mainly targets the evaluation of a feature set through fusion of three different objectives: first, accuracy of the feature set; second, the total number of features in the feature set; and third, training time of the feature set. The evaluation step uses a "normalized score of mixed" (NSOM) method and ranks based on the set having maximum NSOM value. In order to reduce the bias at the fusion stage among decisions, a number of set operations like "union" and "intersection" of feature set elements are carried out in the step of acquiring the feature set as a decision or outcome at last stage. The results obtained on three different datasets like UNSW-NB15, CIC-IDS2017, and CSE-CIC-IDS2018 of networks attack type classifications showed the excellent working capability of the newly introduced ensemble method when used along with a "random forest" classifier method. The accuracy was found out to be 98.37%, 99.92%, and 99.04% on UNSW-NB15, CIC-IDS2017, and CSE-CIC-IDS2018 respectively.

The approach presented in [129], the method proposed is an integration strategy of output characteristics extracted using different FS techniques using the Multiple Criteria Decision-Making (MCDM) methodological framework. The first set of features was extracted using four filter methods. The feature subsets were evaluated using Evaluation Based on Distance from Average Solution (EDAS), an MCDM method, on both the positive distance of the averages and the negative distance of the averages. The experiment has proved the proposed method is able to efficiently

reduce the feature set extracted using FS methods with only a slight loss of accuracy.

In another study, they suggested an ensemble approach incorporating feature-subsets extracted using different filtering approaches via feature-class and feature-feature mutual informations, as explained in Hoque et al. (2018) [130]. This approach is a very efficient combination of feature search and feature relevance calculation. For testing their developed model performance, they used 19 datasets, out of which two datasets are specifically used for categorizing varying network intrusion types in the field of cybersecurity. It is significant that the accuracy level of about 98% is achieved by NSL-KDD datasets when tested using their ensemble approach.

A new ensemble model proposed by [131] combines two subsets of features, one of which is obtained using information gain, while the other is obtained using gain ratio criteria. The performance of the model is thoroughly tested using the IoTID20 dataset [132] and the NSL-KDD dataset. The outcome of the study reveals that the model performs exceptionally well in terms of classification accuracy in both datasets, beating many state-of-the-art IDS. In a similar context, a new model is also proposed with the aim of improving FS confidence using an ensemble approach that involves multiple filter feature subsets [133]. The proposed ensemble method consists of three correlation filters, namely Kendall, Spearman, and Pearson, followed by a nomination step. However, experimental results performed using four datasets indicate that the ensemble approach decreases significantly the size of the tested dataset with a confidence factor of 66%, while maintaining a significantly high F-measure value for properly identifying various types of intrusions.

In addition, a new strategy put forth by [125] incorporates an ensemble FS technique with a deep neural network with a recurrent architecture (RNN), which uses Tasmanian Devil Optimization (TDO) for attack detection in a cloud network for analyzing the performance of attacks such as DDoS, Probe, user-to-root, and Remote-to-Local attacks. The sub-feature can be generated based on either a wrapper technique, filter technique, or integrated FS technique based on the information gain. An information gain ratio is calculated for the importance and relevancy of the set of sub-features.

2.7.3 Collaborative Approach for Lightweight IDS

With a collaborative paradigm, the devices interact within a collaborative environment to improve the intrusion detection capabilities of the network. IoT devices interact in a collaborative manner; hence, they share the duty of recognizing anomalies in a network through multiple nodes. The collaborative method used in IoT-based IDS maximizes resource usage, promotes efficiency of detection procedures, as well as accuracy of anomaly detection [134].

Furthermore, through this method, the IoT devices work together or connect to distribute the process of detecting anomalies on the network. This method is specifically created for detecting IDS on an IoT network and optimizes resource use, increases the speed of detection, and increases the accuracy of detecting anomalies [134]. Moreover, this collective strategy eliminates the burden on the individual IoT device since it distributes all the processes, thus greatly lowering the burden on each device. It is necessary to distribute all the processes related to computations

to prevent resource exhaustion on the device.

Moreover, this collaborative-based technique also facilitates a concurrent inspection of network traffic. In contrast to using a single device for examining all network traffic data, separate devices will examine separate segments of this traffic at the same time. Such concurrent execution is a faster means of examining network traffic data. Moreover, combining the processing capabilities of several devices increases accuracy in identifying anomalies and improving detection capabilities based on their analysis [134].

Arshad et al. [134] have designed a collaborative intrusion detection system called COLIDE and addressed this issue for resource-limited IoT devices. COLIDE follows a two-tier attack detection strategy, which includes device-level and edge router-level attack detection. On the device level, the system needs a lightweight IDS component with three parts: (i) a network monitor, (ii) a system monitor, and (iii) a detection engine. These monitors acquire relevant data which will be processed for possible intrusions using attack signatures. On the edge router level, there are three components beyond the above-mentioned modules: (i) the Alert Collector (AC), which collects alerts produced by the network and system monitors at the device level. (ii) The Correlation Agent (CA), which recognizes patterns of intrusions with a main objective of finding patterns concerning sequences of events towards specific vulnerabilities. (iii) The Detection Agent (DA), which processes and correlates alerts collected by the Alert Collector and Correlation Agent. The COLIDE system thoroughly stresses a collaborative technique for detecting intrusions in an IoT network, and a crucial function has been assigned to the edge router

for detecting and countering threats effectively.

[135] proposed a lightweight approach amalgamating ML, IoT, and cloud computing to facilitate an economical distributed computing approach utilizing Raspberry Pi. The solution explains the requirements of hybrid architecture design by showing core functionalities. IoT nodes support contemporary ML applications, adaptive management of cloud resource lifecycles, allocation of cloud resources via IoT edge devices, and minimizing cloud resource costs. The proposed architecture is divided into three main parts: client-side, server-side, and elastic resources. The case study on the IoT-Pi system proves the effectiveness of the system in minimizing idle AWS cloud instances on Raspberry Pi 4 Model B with more than 70% accuracy in ML prediction. The case study illustrates the applicability of utilizing IoT devices for cloud resource management with the help of ML approaches. However, the system uses a serial execution environment, hindering concurrent resource utilization with other system components. Based on this characteristic, it may be inferred that, although effective in cloud resource management for ML applications, challenges may arise related to resource utilization and coordination.

[136] also developed a distributed and lightweight IDS system with fog and cloud computing architecture. The fog computing part comprises detection, preprocessing of features, applying mitigation techniques, and model learning, while the cloud part mainly deals with identifying different types of attacks through classification techniques. Their proposed model incorporates two efficient classification models: Variational Autoencoder (VAE) and Multi-Layer Perceptron (MLP) models. The VAE model enables dimensionality reduction and extraction of valuable

features from the dataset, while the MLP model distinguishes different types of attacks. Analysis reveals a high level of accuracy and efficiency, although the study does not evaluate the resources demanded during model training and testing, implying a need to investigate the computational requirements for implementing the proposed model for the IDS system.

[137] suggested the E-Spion IDS architecture that has a client-side IoT device as well as the server-side edge server. The IoT device is used for extracting system logs that are sent to the edge server for performing resource-intensive operations like parsing the system log files, extracting the feature set corresponding to the system logs, authenticating system log files, training classifiers, as well as the execution of classifiers. IoT device loading is minimized with the E-Spion model. Multiple classifiers were compared for attack detection. Random Forest classifiers showed the maximum attack detection success rates. There is an increase in system as well as network overhead due to frequent transfer of system logs between the IoT device and the edge server.

The methods used during the design of lightweight intrusion detection systems for resource-limited IoT networks are illustrated in Table 2.8. Because of the reduced processing capabilities found on IoT devices, the need for designing efficient and resource-friendly solutions for IDS cannot be overemphasized. As previously discussed, an anomaly-driven IDS is built with ML methodologies, yet these designs consume high resources for effective operation. Consequently, the strategy followed by researchers designing such IoT solutions has incorporated FS and EFS techniques into their designs.

Table 2.8: Review of state-of-the-art studies for lightweight IDS

Paper	Year	Dataset	ML Algorithm(s)	Accuracy	Approach	System Specs		Time (ms)	
						Memory	CPU	Testing	Training
[110]	2023	TON_IoT & BOT-IoT	NB, SVM, LR, ANN, LSTM	99%	Feature Selection	-	-	-	-
[111]	2023	KDD-CUP-1999, BOT-IoT & NBaIoT-2021	Stochastic gradient descent classifier	94%	Feature Selection	-	-	2.5	4529.812
[126]	2022	UNSW-NB15, CIC-IDS2017 & CIC-IDS2018	RF, DT	99.9%	Ensemble FS	128GB	3.6GHz	243	12
[129]	2022	QSAR, MUSJv2, LSVT voice	DT	91%	Ensemble FS	-	-	-	-
[135]	2022	peer-py dataset	SVM, KNN, LR, MLP	70%	Collaborative	-	-	173000	43000
[112]	2022	NSL-KDD, CICIDS2017	B-Stacking classifiers	98.50%	Feature Selection	8GB	2.9GHz	20	280
[136]	2022	IoT23 & IoT Network intrusion dataset	Variation AutoEncoder MLP	99%	Collaborative	-	-	-	-
[114]	2021	UNSW-NB15	SVM	98%	Feature Selection	-	-	-	-
[115]	2021	BoT-IoT	KNN, SVM, DT, RF, NN, NB, QDA	90%	Feature Selection	-	-	426	-
[116]	2021	IoTID20	SVM, KNN, MLP, DT, RF, LDA	98%	Feature Selection	4GB	-	80	-
[117]	2021	ADFA-LD	MLP	96%	Feature Selection	-	4.4GHz	4.404	-
[60]	2020	NSL-KDD, KDD-CUP-1999	RF	99.9%	Feature Selection	-	-	1.78	0.28
[137]	2020	Self-generated	NB, LR, KNN, RF, DT, ANN	99%	Collaborative	-	-	-	-
[118]	2020	AWID	SVM	98.22%	Feature Selection	-	-	-	299.97
[119]	2020	-	DT	83%	Feature Selection	-	-	-	-
[120]	2020	Self-generated	RF, DT, SVM, BPNN	99.70%	Feature Selection	-	-	30000	113000
[121]	2020	Self-generated	ANN	99.90%	Feature Selection	-	-	1.7	2.7
[138]	2019	-	Fuzzy logic	80%	Feature Selection	-	-	-	-
[123]	2019	Self-generated	J48	100%	Feature Selection	128GB	2.6GHz	0.0351	2.2813
[124]	2019	CICIDS2017	SVM	98.03%	Feature Selection	-	-	2.2813	208.9063
[139]	2019	UNSW-NB15	DT	99%	Feature Selection	-	-	8750	80980
[134]	2019	-	-	-	Collaborative	128GB	-	-	-
[130]	2018	UCI	DT, RF, KNN, SVM	95%	Ensemble FS	12GB	2.26GHz	-	-

As made evident by the analysis performed, “Feature Selection” stands out as the most popular alternative among various techniques in implementing “lightweight IDS” systems in comparison to other techniques like the collaborative one. Unfortunately, with a dynamically evolving research environment, there has been a sharp transition to the newly emerging “extended FS” techniques for efficiently developing “lightweight IDS” in the “IoT” environment because of its “effective” potential to “yield” better “classification” models.

In addition, it can be seen that FS and EFS strategies play a role in improving the accuracy rate as well as optimizing the use of CPU, memory, and the time taken to train and test the classification model altogether. However, it can be seen that the collaborative method uses the availability of cloud and edge-layer infrastructure to deal with resource-demanding tasks associated with IDS. However, a handful of research works, [135], [134], and [136], have utilized the collaborative method to design the IDS because it requires a proper communication mechanism to safely implement the trained model from the cloud/edge to the IoT device. Moreover, Table 2.8 lists the techniques utilized to design the lightweight IDS systems for the IoT device based on the use of FS, EFS, and collaborative methods.

2.8 Self-Healing Mechanisms for Automated Recovery

Self-healing can be defined as a concept that has garnered widespread adoption in many different fields, such as computing, material science, and biologic systems [140]. The core idea involves systems and materials with the capability to self-

assess and heal themselves without human interaction [140]. Self-healing involves the autonomous healing process that entails the self-assessment and correction of system faults without human assistance [140]. Such a healing process involves either temporary and enduring solutions for continued system functionality [140]. The main attraction for self-healing systems involves their possible beneficial effects for improved reliability [140].

2.8.1 Key Steps of Self-Healing System

A set of basic processes is responsible for self-healing, facilitating efficient error detection, diagnosis, containing, and correction, described by [141]. These steps typically consist of the following:

Fault Detection: The system is always running in the background in order to detect anomalies in its surroundings that could be an indication that a fault exists. Real-time analysis is central in this component.

Fault Diagnosis: On the detection of a potential fault, the system examines the data collected to determine the nature of the fault. The use of algorithms in the data resulting from fault detection identifies the cause of the fault.

Fault Isolation: The faulty components and processes will be isolated and will be prevented from propagating to the whole system. These might involve mechanisms that will compartmentalize a particular area and will temporarily disable some functions.

Fault Recovery: After isolation, the system initiates fault recovery tech-

niques. These may include the use of predetermined corrective measures, redundancy, or adherence to system recovery protocols aimed at restoring normal system functioning.

Self-Adaptation: It changes itself based on experiential knowledge obtained through fault recovery. It involves modifying fault recovery approaches, reconfiguring components, or changing parameters to increase resilience based on fault recovery experiences.

Fault Prevention: Preventive actions based on knowledge obtained from faulty and recovery experiences are applied. This includes modifying the system policies and improving the monitoring and diagnosis approaches to enhance robustness.

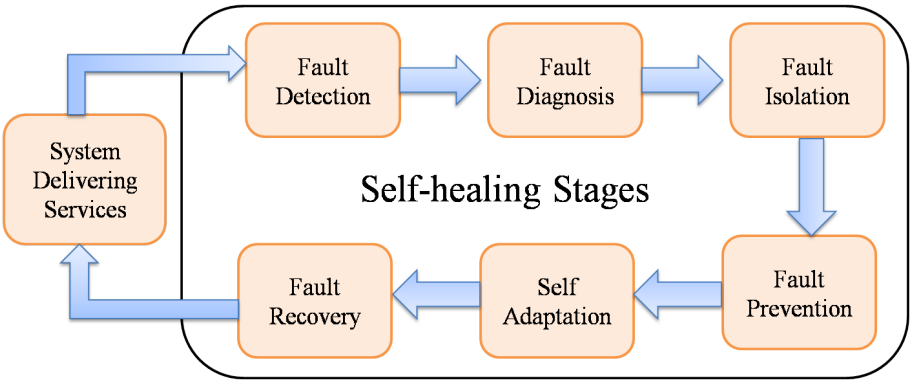


Figure 2.4: Stages in Self-healing system

The phases depicted above in Figure 2.4 work together for the system's ability to independently identify, control, and recover from any faults, thus maintaining high levels of reliability and efficiency, built on the work described by [142].

Such processes provide higher robustness in a system because they ensure that

failures are resolved in an efficient manner and will not cause system deterioration or unexpected failures resulting in poor performance. Thus, this solution increases user satisfaction due to dependability in service provision.

2.8.2 Theories in self-healing

The theories associated with self-healing provide a substantial area of research; it encompasses the attempt to interpret and further reinforce the fundamental principles under which self-healing functions within systems. The theories not only try to identify the dominant characteristics necessary for self-healing but attempt a paradigm shift by placing self-healing algorithms within a scientific spectrum. By examining the inherent patterns between self-healing and associated domains like system theory, control theories, and biology, a multidisciplinary area necessary for the development of self-healing methodologies is pursued [143].

2.8.2.1 Negative and Positive Selection

Negative and positive selections are vital processes in the immune system whereby harmful elements are removed and healthy ones restored. These processes and theories can be related and used in self-healing systems whereby a system automatically heals itself in the event it gets compromised or damaged in some way. In self-healing systems and technology, the application of negative and positive selections in the immune system can be used in creating a system with mechanisms to remove harmful elements and restore/promote healthy ones in a system. In terms of biological theories and processes in the immune system whereby a system creates a means to

fight off harmful elements in a system or within its surroundings, biological theories would define negative selections as “the recognition and elimination of cells that recognize self-components to prevent autoimmunity by reducing the probability of normal body cells being targeted as foreign by the immune system” (JohnPhillips 2023). Conversely, positive selections would be defined as “the recognition and selection of cells to recognize foreign components to enable them to detect and target harmful invaders or other dangers” in a system (JohnPhillips 2023). The theory of self-healing in computers and technology would, therefore, be based on the immune system’s processes whereby a system creates a mechanism to eliminate potential threats and also a system to detect and promote elements within a system or its surroundings to improve its defensive measures against harmful invaders/pathogens in a system or within its environment [143].

Hence, from the field of computer science, an application or solution regarding intrusion detection could be developed by applying the concept of genetic algorithms. These algorithms work by detecting anomalies and intrusions in systems by simulating evolutionary processes. After detecting an intrusion in the system, the capability derived from the self-healing nature of the algorithm should then work towards healing the system from the resulting threat by isolating and removing possible subversive components by negative selection or by modifying and improving system defenses against intrusions by positive selection. The definition belonging to the nature of anomalies and guiding the systematic detection of potential subversive regions in the system is greatly related to the theory based on negative and positive selection. This theory has high importance in the detection process

of subversive components in the system by providing an efficient framework in its application regarding system security. In connection with the primary theory in this work, negative and positive detection principles in the context of anomaly detection in systems are described as follows: Negative detection in the area of anomaly detection could be described by applying the term "non-self" detection, while positive detection would refer to the term "self" detection [144].

The key concept of negative selection, as in Figure 2.5, involves building a pool of “non-self” objects, meaning any entity that does not match a similarity criterion against any existing “self” objects, as proposed by [145]. If a particular entity matches any of the “non-self” objects, such an entity is labeled as “foreign” and automatically rejected. This process ensures that only “self” objects are in operation in the system, thereby eliminating any possible threats or faults. The identification of such anomalies is critical for ensuring the implementation of self-healing functions in a timely manner. The positive selection criterion simplifies the process by one step, where a particular entity is checked for similarity against the existing “self” pool but not against the “non-self” pool; hence, failure of which leads to an automatic rejection of the entity into the system, ensuring that only “self” is in operation in the system.

According to D’haeseleer [144], negative selection represents an exemplary form of an efficient immune system because it does not need prior knowledge about intrusions. Negative selection is essentially an all-purpose anomaly detection algorithm; it is self-learning and constantly evolves to become an ever-changing set of detectors. If the outdated detectors become unnecessary, they will be flushed

away, and new detectors would be created from the current flow of event traffic. This way, it will constantly adapt to the evolving danger. According to Dasgupta [144], despite their differing base models, negative and positive selection algorithms yield analogous results. There is essentially no difference between them in triggering an alarm when an alien entity breaks in; thus, they ensure prompt detection and reaction to the danger.

Thus, it can be said that the concept of self-healing computing based on negative and positive selection functions in the same manner as the immune system does in keeping the system healthy. This way, scientists are able to devise sophisticated self-healing systems that can automatically identify and heal themselves from the attacks.

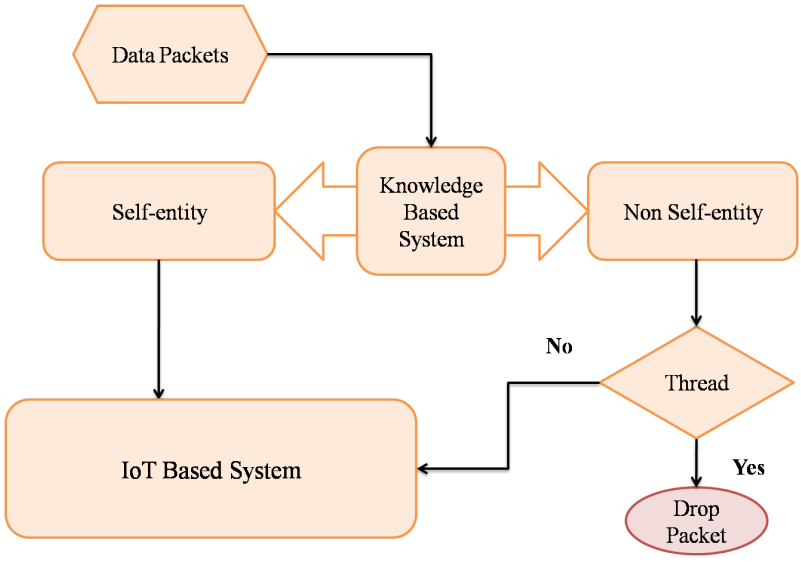


Figure 2.5: Negative and positive selection

2.8.2.2 Danger Theory

Danger theory: There is an opposing theory regarding the activation of immune responses based on danger signals rather than the presence of "self" or "non-self" elements [146]. In this theory, any foreign body encountered in either positive or negative selection is tolerated until a danger cue is perceived. For example, as illustrated in Figure 2.8, a healing response in the immune system is triggered only upon the perception of harmful activities. Based on the severity of the danger cue, a massive attack or a local attack on foreign or local entities is considered.

Indeed, as cited by [144], the notion of biologically inspired danger theory was proposed by Burgess et al. with the aim of detecting dangerous activities on computer systems. Mazinger, adopting the view of [146], argues that the immune system is not capable of distinction between self and non-self, but instead between events with the ability to inflict damage and those lacking such ability. Furthermore, this allows the system to discriminate its reaction according to the actual threat level an event may pose, instead of whether it is self or non-self alone. When the system has complete information about its self-state, its pattern recognition mechanism will also be capable of responding appropriately to events that pose threats.

Thus, danger theory becomes a crucial component of threat detection because it focuses on potential danger rather than the rigid classification of entities. A danger theory-based system would provide a more precise security mechanism for defending against potential threats. A danger theory-based IDS implementation described by the self-healing paradigm presented by [144] would calculate the anomaly score of

each event occurring within the system. This model primarily focuses on three essential metrics that define event analysis: the event type, anomaly, and danger value.

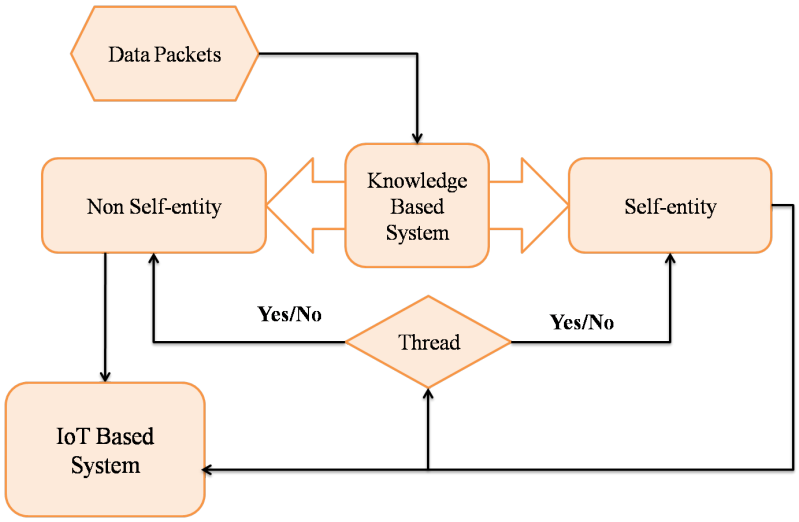


Figure 2.6: Danger theory to optimize self-healing

The EV measure is based on predefined criteria or cluster analysis for events and helps provide a notion of event type definition for understanding its significance within the system as a whole. The AV value measures the level of event anomaly based on computations related to entities other than itself and helps determine a level of departure from normal event activities. The DV increases as a result of its relation to abnormal or dangerous signals associated with the event and signifies the level of risk or threat associated with it. These three values are combined to compute the Threat Total Value (TV) and signify the potential level of harm or contribution to system failure as a consequence of the event's occurrence. The TV of the television helps determine its total significance as a threat source associated with each event. System response flow related to dangerous events involves three

key processes:

*New Event Analysis:*As soon as a new event is observed, it is injected into the system timeline. The system then analyzes patterns which could indicate dangerous behavior. This step involves evaluating the new event within the existing history of data for potential threats.

Danger Signal Processing: After a danger signal has been indicated, it is assessed to see if any patterns in event data indicate a relation to this danger signal and act upon this relation with the intention of reducing any threats.

Warning Signal Processing: In case a warning signal is detected from external hosts concerning a possible threat and a corresponding chain of events, the system checks the system timeline to determine if a similar chain of events has occurred. This ensures that the system is adequately equipped to deal with a common threat.

With the integration of these phases, the self-healing system based on danger theory increases its ability to defend against possible attacks, and thus its ability to maintain system integrity increases.

2.8.3 Literature Survey on IDS with Self-healing Capability

The application of self-healing in IDS addresses the issue of zero day attacks because it enables a system capable of autonomous detection, diagnosis, and healing from any kind of disturbance. A self-healing IDS becomes capable of working in a real-world environment with normal operations unaffected by the intrusion event. The reason why a self-healing IDS has this ability is based on the monitoring of the

system, identifying operational characteristics of a system that refer to a break-in, followed by autonomous actions that can reduce the resultant threats. Applying self-healing in IDS systems not only improves their operations, it is necessary in today's world of cybersecurity threats. As more complex approaches emerge from attackers every day, cybersecurity systems need to align with these developments. A self-healing IDS system indicates the future of cybersecurity in terms of providing a proactive defense mechanism that moves beyond the conventional approach. [147].

Within this context, existing research has unveiled various approaches with a focus on improving security solution resilience through self-healing processes. For instance, [148] in 2024 highlight the use of bio-inspired approaches entitled Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and Genetic Algorithms (GA) in improving sensor network resilience, adaptability, and efficiency, based on a reference in [149]. ACO aims at improving routing and healing through ant colony optimization, PSO aims at improving management and self-organization through particle swirling, and GA aims at improving sensor location and healing processes through genetic evolution. Results clearly show that the proposed technique in [148] in 2024 realized a precision level of 95.3%. Nevertheless, bio-inspired models, especially GA, are characterized by substantial complexity and may limit adaptability in megaeuropeana-scale sensor networks due to excessive computations [150]. Furthermore, ACO and PSO models may encounter convergence limitations in particularly expansive search spaces characterized by large-dimensional features and dynamic sensor network scenarios [151].

A recent work [64] introduces a new self-hearing technique for improving

cloud computing infrastructure using Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) networks for predicting cloud computing system failures with a promising accuracy of 95% while maintaining a low false positive rate of 3%. The method includes proactive fault processing through resource allocation/deallocation, workload migration, and backup job execution. This synergy of cloud computing with reinforcement learning helps to make cloud computing more reliable and efficient. On the other hand, retraining of RNNs/LSTMs is an issue for large-scale implementation of these techniques because of heavy computations required by these approaches for their implementation, which is not feasible for resource-limited IoT environments.

Another study [152] introduces the concept of SHARP-Net, the self-healing and attack-resilient phasor measurement unit (PMU) network that makes up the smart grid infrastructure. There are three major components in the SHARP-Net system. They include the intrusion detection system (IDS) that uses synchrophasor network log data in detecting possible cyberattacks. After the detection of possible attacks through the generation of alerts by the IDS, the alerts are transmitted to the IMS through the Alert Management System (AMS). The work shows that the system has the capability to detect cyberattacks as well as mitigate them successfully within the shortest possible time. However, the system uses the study of predefined rules extensively in the detection of attacks. This makes the system less effective in the detection of new attacks.

The paper "Hybrid Intrusion Detection System Using Signature and Anomaly Detection Methods" by [28] offers a hybrid approach for improving Cybersecurity

using both signature-based and anomaly-based detection approaches. The suggested self-healing mechanism is divided into three phases: signature-based detection, anomaly detection, and generation of signature. Signature-based detection is very efficient in recognizing known threats, thereby relieving the anomaly detection phase from extra burden. Anomaly detected from LSTM-RNN is further validated; attributes are converted into new signatures for continuous learning and adaptation without requiring human involvement. It was tested using modern datasets: UNSW-NB15 and ADFA-LD, which cover modern-day attack scenarios. C5 Classifier yielded 97% detection accuracy with 8% false positives, while LSTM-RNN yielded 90% detection accuracy with 17% false positives. Some drawbacks associated with this approach include lack of support for detection related to stealth or 'obfuscation' type attacks that can easily circumvent defense systems. Also, 17% false positives associated with anomaly-based detection can lead to wasteful consumption of resources.

The study [153] proposed framework is developed that uses artificial intelligence (AI) and ML techniques to counter the effects of ransomware attacks by implementing dynamic network segmentation, real-time anomaly identification, as well as AI-based recovery strategies. The framework is capable of isolating the targeted network segments in less than 15 seconds, identifies ransomware in 3.1 seconds, and achieves a success rate of more than 90%, rising to nearly 99% in isolation success rate for repeated attacks. However, the proposed framework has some limitations in terms of scalability for large network configurations, susceptibility to ransomware that attacks backup infrastructure, as well as challenges posed by polymorphic ran-

somware. The proposed work does not provide complete quantitative assessment outcomes.

Another study [142] the research work proposes a framework that helps Internet of Things (IoT) devices identify faults and cyberattacks, as well as mitigate them. The framework consists of three interlinked modules: (1) a Host-Based Intrusion Detection System (HIDS) module for anomaly detection, (2) a Health-Monitoring module for performance measurement of the device, and (3) an Autoremediation module that performs corrective measures using pre-trained neural networks and evolutionary algorithms. The model makes use of collaborative learning, which enables a network of IoT devices to learn efficiently by sharing information about device health using blockchain technology.

The proposed framework was tested on IoTs in scenarios such as Denial of Service (DoS) and process exploitation, and it showed a large amount of adaptability. The accuracy of remediation strategies improved as the number of gadgets in the network increased. However, this study has some limitations in relying only on Host-based IDS (HIDS) in real-world attacks that are network-based and multi-vector in nature, such as DDoS attacks. Furthermore, the results were not presented in terms of quantifying the system's effectiveness. Likewise, Abdulrazak et al. in [154], proposed a system that could make IoTs heal themselves by incorporating architectural approaches to address networking issues, software issues, hardware issues, interaction issues, and security issues comprehensively. This proposed system utilized the AMI system along with a multi-layered approach of Device, Fog, and Cloud that was supplemented by autonomic computing principles and virtualization.

It had three key elements: Sensing Approach (SenS), that was utilized to identify IoTs; Awareness of Issues (AwaR), that was utilized to detect anomalies in IoTs; and Responsibility and Actions (ReAct), that was utilized to make remediations with autonomy. The results showed that there was a large improvement in its dependability, in that the data range enhanced from 4% to 75%, along with an accuracy rate of 99% in self-healing. The authors considered some threshold levels that were not tested in all scenarios comprehensively; these included a threshold of 50% battery level. The authors should have included customized thresholds to adapt to various levels of various IoTs in various environmental situations to avoid poor results.

In study [155], the authors propose Penta.py, an agent-based network model with a self-healing mechanism to heal the network automatically upon the identification of the vulnerability in the system. By incorporating agent-based network management and a self-healing model, this system has the ability to automate vulnerability scanning and its related remediations without any intervention in the system by humans. Moreover, this system uses Nessus vulnerability scanning and a centralized database to store network settings and respective patches for vulnerability scanning and management. Penta.py has the capability to detect the missing patch and download the necessary update and apply it automatically to the system in question to decrease the system's downtime and associated threats. The suggested technique has a restricted scope and only focuses on missing patching in Windows 7 environments; therefore, it limits its usage in other operating systems and related vulnerabilities. The research does not explore the preventive and/or detective stage

in detail either.

Another study [156] introduces a framework that improves resilience during distributed IoT process tasks through the use of MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge). Based on the PROtEUS workflow engine, this concept combines a self-healing mechanism capable of monitoring the status of distributed IoT devices and processes for diagnosis and planning purposes. As a means of improving system resilience during distributed process tasks, this concept relays tasks to other devices as appropriate for enhanced failure tolerance during distributed process task executions. However, a major flaw associated with this concept lies in the increased requirement for super-peer nodes.

Authors in [157], in this study, a new paradigm based on the human immunity system has been proposed for defending enterprise networks from botnet-based cyber attacks. The proposed self-healing mechanism consists of a server for system states analysis and management, a set of agents for infection detection and healing in individual systems, and a quarantine network that can segregate systems beyond repair from the rest of the network, thus significantly suppressing malware diffusion. Although taking complete advantages of the managed setting of the current enterprise networks, the proposed paradigm can automatically identify, heal, and quarantine the attacked systems without any human intervention. Experiments in this research work used IRC-based botnets named RxBots and HTTP-based WarBots for verifying the efficacy of the proposed paradigm in healing attacked systems and protecting network assets. One of the main issues in this study work is that the new paradigm relies on certain enterprise settings that can limit their universality

in less managed settings of general Internet networks.

Another study [158], biologically inspired approach is described that integrates an Artificial Immune System for intrusion detection with a self-healing process. Inspired by idiotypic networks of biological immune systems, this approach promotes dynamic and adaptive defense against network threats through distinguishing learned characteristics of 'self' (normal behavior) and 'non-self' (anomalous behavior). For detected intrusions, a self-healing process is capable of autonomously diagnosing an error, locating damage, and taking appropriate recovery measures, for example, repairing parameters and evolving a topological network structure. The article by [158] introduces an interesting method but provides no quantitative analysis to support their proposed strategy.

2.9 Research Gaps and Emerging Challenges in DoS/DDoS Attack Detection for Resource-Constrained IoT Devices

- **limitations with filter-based FS in designing lightweight IDS:** From the literature review covered under section 2.7, it is clear that FS is found to have been used widely in the design of lightweight intrusion detection systems for resource-constrained IoT devices. FS using filters is widely used for selecting highly correlated features from a pool of available features in a dataset for classification tasks. Meanwhile, filter-based FS is also known for efficiency gains in lowering both training time as well as test time with reduced resource cost. However, using FS with filter-based techniques as a stand-alone technique might limit adaptability and efficiency of performance of the IDS

model since filter-based FS requires pre-defined statistical thresholds to evaluate feature relevance. Although using statistical thresholds for FS might result in efficiency gains for faster FS, statistical thresholds might not capture subtle underlying patterns adequately to avoid-feature exclusion with potential relevance to address specific attack detection needs for more generalizable outcomes.

- **Challenges in Dataset Integrity** The training data of ML models usually consists of a huge set of features. The presence of features directly relates to the usage of computational resources. Moreover, these datasets in the IoT era are often artificially generated instead of being taken from real-world scenarios. Therefore, the IDS models are trained in environments that may not resemble real-world settings. Furthermore, these datasets lack sufficiency for the task of zero day attack detection. Therefore, there exists a great requirement for real-world datasets that can increase the efficiency of IDS models.

- **Recent DoS/DDoS attacks**

  The discussion shown in Section 2.2.3 clearly explains how DoS/DDoS attacks affect the services provided for lawful users. However, due to the complexities involved with current DoS/DDoS attacks, it is clear that current security measures lack efficacy in detecting these attacks. Specifically, the current methods may lack effectiveness, especially in detecting advanced attack methods like zero-day attacks, which usually exploit unknown vulnerabilities. Hence, the development of more effective security measures is necessary to facilitate the detection of advanced DoS/DDoS attack methods.

- **IDS gap adaptive approach for detection of new attacks**The process design for IDS that uses ML algorithms mandates intensive resource support for accommodating training and testing processes. In the context of securing resource-constrained Internet of Things environments against constantly emerging threats, an integral need exists for adaptive security solutions that are resource-effective and efficient. These solutions should be able to adapt dynamically to emerging threats.

- **Resource Efficient Self-healing Approach** The development of resource-efficient algorithms that can be applied in scenarios with limited processing power and energy. Modern self-healing techniques often adopt computationally hungry models like XGBoosting and LSTM, which can pose a considerable processing workload on the system, especially in massive IoT networks.

2.10 Summary

This chapter delivers a comprehensive review on DoS/DDoS attacks, focusing on their variants as well as examples of their history over the years as a means of understanding their development and influence. In addition to that, this chapter willalso delve into the impact these attacks may cause on Internet of Things (IoT). Afterward, this chapter will discuss the topic of IDS, focusing on their categories as well as the adoption of ML algorithms into IDS systems. Additionally, this chapter delivers a comprehensive review on available data sets focusing not only on their latest updates but rather on the latest state-of-the-art IDS technology available today.

In addition, a critical analysis is also provided for the available methods designed for lightweight security solutions to counter DoS and DDoS threats by evaluating the efficiency and effectiveness of the methods being utilized today. This chapter further explores self-healing methods and their theories and implementations, besides scrutinizing how these methods are utilized inside the frameworks of IDS to enhance resilience and adaptability to any systems being threatened by DoS/DDoS. The final section does address the limitations and open challenges encapsulated inside the literature to be overcome to further enhance security solutions towards DoS/DDoS threats.

CHAPTER 3

Ensemble Feature Selection Approach for Lightweight IDS

3.1 Introduction

The growing cyber-attacks and IoT device resource limitations highlight the need for ML-enabled lightweight security solutions to protect IoT systems from emerging attacks. The literature survey conducted in Chapter 2 highlights that, except for a few existing ML-based lightweight IDS, most solutions encounter significant challenges, such as high false alarm rates and substantial resource requirements for execution. These challenges limit the effectiveness and applicability of existing IDS in resource-constrained IoT environments. Addressing these issues is essential to developing efficient security mechanisms that provide robust protection without overburdening IoT devices. This chapter presents the mechanism for designing a lightweight IDS tailored to protect IoT systems. The proposed mechanism focuses on designing an

IDS that can effectively detect and respond to emerging cyber-attacks while being optimized to operate efficiently with the limited computational power of IoT devices. An effective security solution ensures robust protection while minimizing computational overhead, thereby preserving the functionality of IoT systems.

3.2 Ensemble Feature Selection

The classical FS such as wrapper, filter, and embedded techniques, are very efficient in solving the problem of dimensionality reduction. This is especially true when it comes to the analysis of network traffic, where FS can be utilized to remove features that are irrelevant or less information-bearing.

The wrapper approach generally results in a higher accuracy rate by assessing the merit of feature subsets based upon their accuracy in a model. This method uses a machine learning technique in which a subset of features is selected iteratively, followed by an estimate of the impact of features on accuracy rates. The wrapper approach requires a lot of computation [159], which makes it impractical for use in large-scale experimentation. Overfitting is also a major problem in the wrapper approach as the selected feature subsets are repeatedly tested on the same data [99].

Taking into consideration the resource constraints that exist inherently within IoT networks and the goal of optimizing the response efficiency of the IDS component within them, the proposed work adopts the use of Filter techniques for the purpose of FS. Such techniques are scalable and efficient for handling high dimensional data while keeping the computational complexity under check [125].

- **Bias in Selection:** Personal biases may arise in FS because each method uses its own set of criteria for evaluation. As a result, the selected features in one method may vary widely from the selected features in another method.
- **Computational Demand of Wrappers:** The wrapper method in FS often involves the use of a strong classifier which could perform very well for a particular dataset and a given problem domain. But it could be very computer-intensive when dealing with a big dataset.
- **Suboptimal Feature Sets:** Using only one FS technique might cause the evolution to have suboptimal feature sets because of the vulnerability to local optima problem. This problem gets worse in a scenario with multiple fitness functions or large search spaces.

As such, the research community is moving forward with an improved solution known as "Ensemble Feature Selection." This is because "ensemble" refers to a set of approaches working together for common goals. Within the paradigm established for FS an ensemble methodology brings multiple approaches together. Every approach has its own set of FS that rely on specific guidelines; these guidelines are later combined using approaches like voting or scoring. Moreover, this paradigm is still an ongoing innovative field in ML, characterized by the integration of multiple models.

3.3 Proposed Ensemble Feature Selection

This article will give a detailed explanation of the new proposal of the Ensemble Feature Selection (Ensemble FS) method for the Lightweight Intrusion Detection

System (ELIDS), which has been designed and aimed at optimizing the efficiency and accuracy of FS. The essence of this methodology clearly involves pinpointing the most important and prominent features out of a massive dataset. The reason for the development of the Ensemble FS methodology can be attributed to the need for reducing the size of massive datasets without sacrificing the key features required for precise IDS systems. Internet of Things (IoT) technology has restricted capabilities and necessitates an IDS with the capability to function with optimal resource utilization. Collectively, conventional FS systems, whether wrapper methods, filters, or included systems, fail to merge the capabilities of optimal computing with ID efficacy.

The Ensemble FS technique reduces these difficulties since it uses several FS techniques at the same time. This reduces the dependency to a single FS technique since their strengths will be utilized during the aggregation process. The combination of diverse results from various techniques is therefore effectively done to produce a better quality of results/FS through the synthesis of strengths from different techniques, as shown in Figure 3.1. ELIDS system incorporates various stages including Step I: Data Preprocessing, Step II: Ensemble FS, Step III: Model Training, and Step IV: Performance Evaluation Parameters. ELIDS aims to create a lightweight FS process to help promote the design and development of more efficient IDS systems concerning both performance and resource usage.

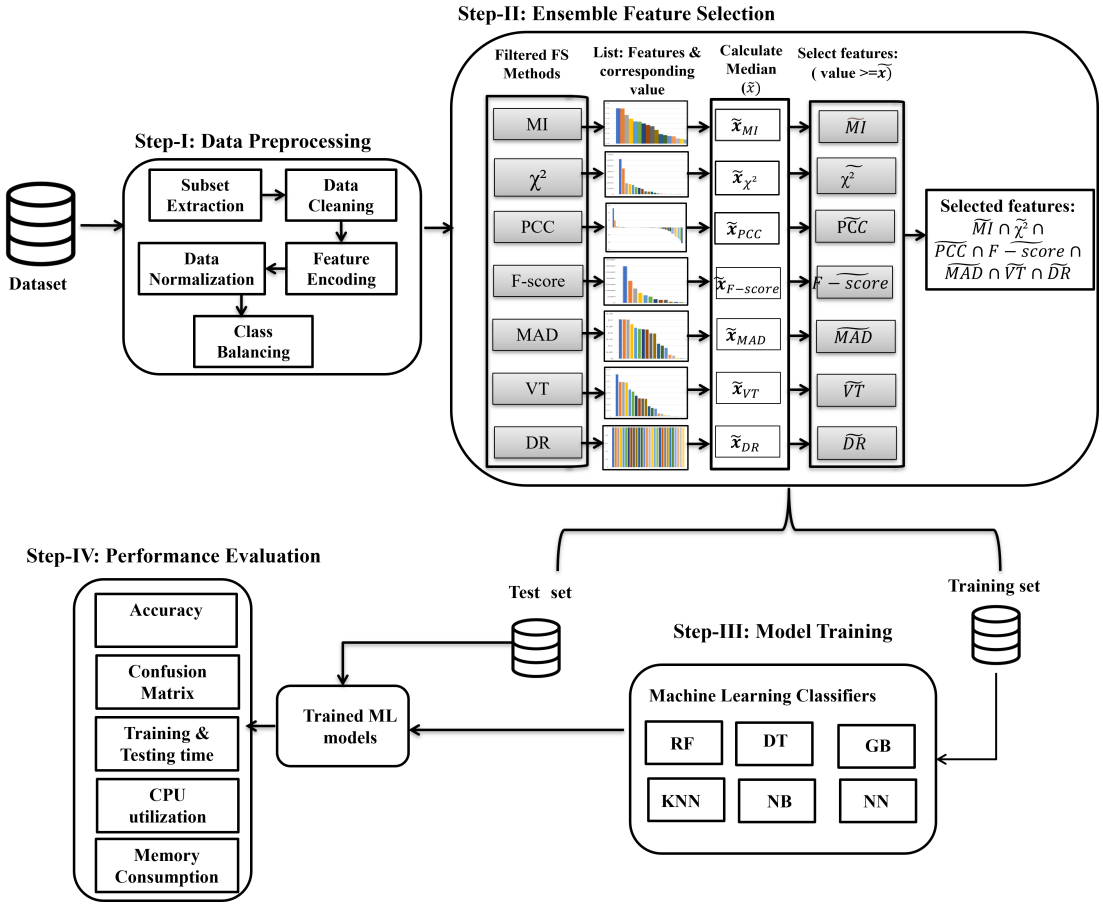


Figure 3.1: Proposed methodology for ELIDS

3.3.1 Data Preprocessing

The process of preprocessing a data set is carried out through five different sub-steps, including subset extraction, data cleaning, feature encoding, data normalization, and class balancing. This is very essential since it prepares the data for learning purposes.

- **Subset Extraction:** As the main research objective in this work revolves around identifying DoS and DDoS attacks, great importance has been assigned to data sampling. Subsets can be extracted by means of simple queries that can yield data. In this research work, a programming language named "Python" has been used for querying the data in search of all records in which the "type" column refers to any of the following values - "DoS," "DDoS," and "Normal." The "Pandas" library in Python has been used in this research work for filtering data based on the above criteria and fetching data pertaining to DoS/DDoS attacks as well as normal data that doesn't belong to any type of attack.
- **Data Cleaning:** The data cleaning stage is an essential part of the data preprocessing process as a whole and a pivotal step towards improving the quality and precision of the dataset. During this step, it is ensured that the data fed into the model is valid and uniform, which is a critical requirement for building a robust machine learning model. Eliminating redundant and null data represents one of the primary concerns when dealing with data cleaning. Redundancies can be generated when a particular piece of data is repeated. Such issues may cause biased outputs or give exorbitant importance to some

patterns while ignoring others. On the contrary, null data may result in introducing substantial noise into the dataset, making it difficult for the model to pick out significant patterns.

With the removal of redundant records, missing values, nulls, and discrepancies, the data cleansing process results in a refinement of the quality and consistency of the dataset. Therefore, the model is trained with a more accurate, consistent, and quality-enhanced dataset, which impacts the model's generalization property directly. This is especially important when it comes to the development of intrusion detection systems (IDS), because the separation between malicious and normal data heavily relies upon the data's clarity and correctness, although it holds true for any field being tackled.

- **Feature Encoding:** Once the data is cleaned, the feature encoding can then be used to ensure that categorical variables are in a numeric format that can therefore be used by a machine learning algorithm that relies on numeric data. Since machine learning algorithms would find it a challenge to process categorical variables directly, the need for feature encoding becomes imperative in ensuring that categorical variables can be used in a format that the machine learning algorithm can understand. Feature encoding in this study uses Label Encoding, which assigns a specific number corresponding to the frequency of that particular label. The entire process ensures that the dataset can work well with the machine learning algorithm in a way that enables it to learn from the categorical variables.

- **Data Normalization:** After the feature encoding step, normalization is per-

formed using the min-max scaling algorithm. This algorithm scales the features into a specified range [0.0, 1.0]. This is done because features with large number ranges could end up taking precedence during the model's learning phase. This is more crucial when it comes to machine learning models because most distance-based models like k-Nearest Neighbors and Support Vector Machine (SVM) as well as neural networks (NN) [97] are sensitive to the features' scales. If the normalization step is skipped, it could end up taking priority during the model's goal attainment phase and could result in biased outputs and poor model performance. This step helps ensure that fairness is upheld during the model's operation and helps the model perform its task of producing correct outputs more effectively.

- **Class Balancing:** Given that ELIDS is currently designed for binary classification, where the goal is to distinguish between malicious and benign traffic, a class balancing technique is employed to mitigate the risk of biased model training. In many real-world scenarios, datasets are often imbalanced, meaning one class (typically normal traffic) may have significantly more samples than the other (DoS/DDoS attacks). This imbalance can lead the model to become overly focused on the majority class, reducing its ability to correctly classify instances from the minority class. To address this, class balancing techniques including oversampling of the minority class or undersampling of the majority class, are applied to ensure that both classes are represented in equal proportions. This step is pivotal in enhancing the model's generalizability and ensuring robust performance across both classes, preventing skewed results

that would compromise the detection of rare but critical attack instances.

By carefully executing these preprocessing steps, the dataset is transformed into a refined form that is well-suited for machine learning model training, ultimately leading to more accurate and reliable detection of DoS/DDoS attacks.

3.3.2 Feature Selection Using Ensemble Approach

In this phase, the proposed ensemble FS technique, shown in Figure 3.2, is used to reduce the dimensionality of the dataset by eliminating a large set of features to a more concise and significant subset.

Figure 3.2 illustrates the ensemble technique incorporating seven filter-based FS methods including MI, χ^2, F-Score, PCC, VT, MAD, and DR. As discussed in Chapter 02, filter methods are particularly efficient compared to other FS techniques, such as wrapper and embedded methods, as they rely on mathematical measures to evaluate each feature independently and require minimal computational resources. Moreover, MI, χ^2, F-Score, PCC, VT, MAD, and DR are popular and widely adopted by the research community in the field of FS.

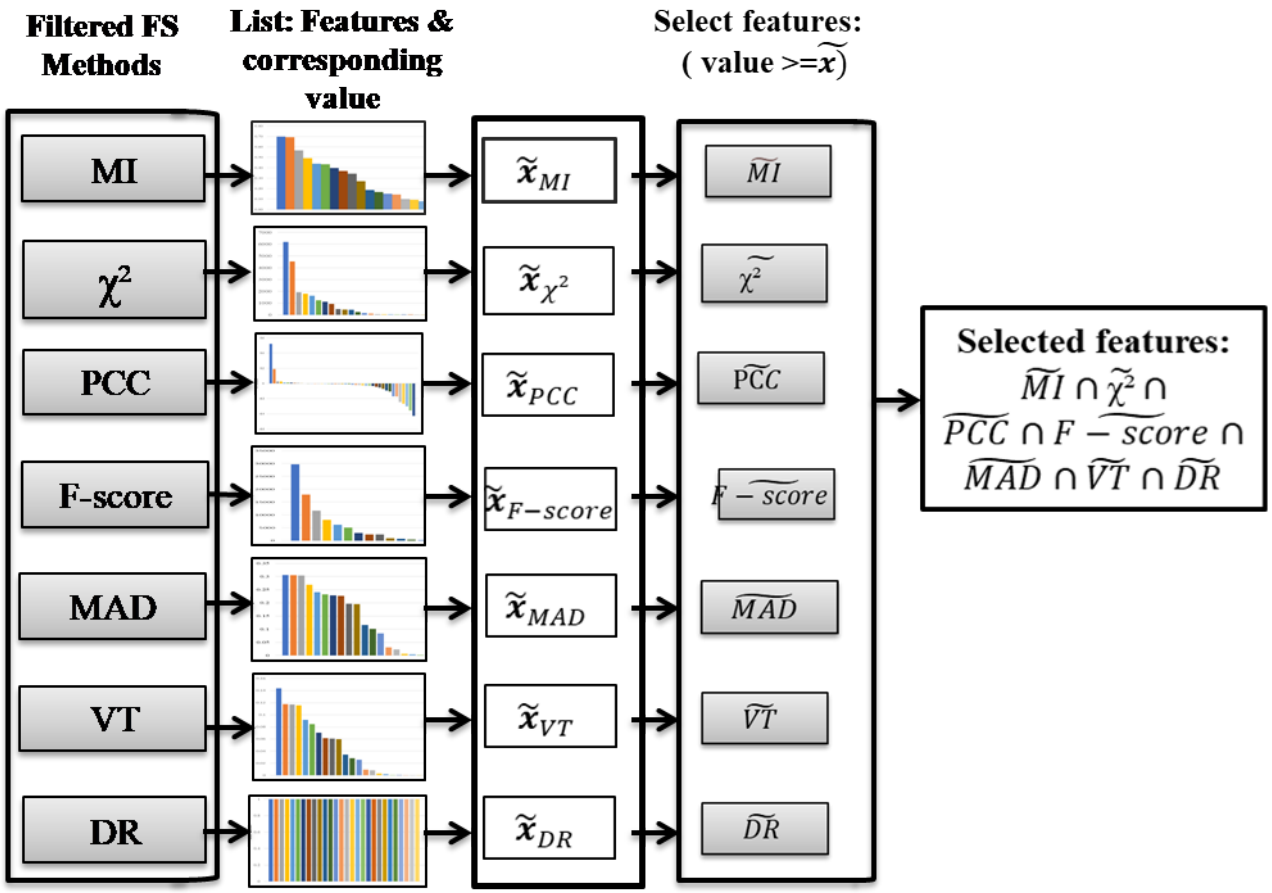


Figure 3.2: Ensemble-based feature selection

For each method, the process is executed independently on the preprocessed dataset. This independent execution ensures that each method's criteria are applied without influence from the other methods, resulting in seven distinct experiments and, consequently, seven unique lists of methods. These lists of feature differ due to each method's distinct selection criteria. After each filter method generates a ranked list, the features are organized in descending order of significance, with the most relevant features positioned at the top.

Each FS method generates a ranked list of features, the median value ($\widetilde{x}$) of each list is computed to establish a threshold for refining the selection. For each list, features with scores or ranks at or above the median threshold are retained, while those below the median are excluded. Let $F_i = \{f_{i1}, f_{i2}, \ldots, f_{in}\}$ be the list of features ranked by a feature selection method i, and let $\widetilde{x}_i$ be the median threshold for that list. The reduced feature set F_i' for each method i is defined in Equation 3.1:

$$F_i' = \{f_{ij} \in F_i \mid f_{ij} \geq \widetilde{x}_i\} \tag{3.1}$$

where F_i' includes only those features f_{ij} (here, i indicates the FS method, and j is the index for each feature) from F_i that have scores or ranks at or above the median threshold $\widetilde{x}_i$, excluding those below the threshold. The process resulted in a reduced feature set for each FS method, keeping only the most significant features. The rationale behind using the median as a threshold is crucial, as it helps to filter features. Unlike the mean or standard deviation, the median is robust against outliers, making it a more reliable measure for threshold determination in datasets

that contain skewed or multimodal distributions. This robustness enhances the FS process by ensuring that the selected features reflect genuine significance rather than being influenced by extreme values.

Once the individual feature lists are reduced based on their respective medians, an additional filtering process is conducted by identifying the intersection of features across all seven FS methods. As given below in Equation 3.2:

$$F_{\text{intersection}} = \bigcap_{i=1}^{7} F_i' \tag{3.2}$$

where $F_{\text{intersection}}$ denotes the set of features that appear in the reduced feature lists $F_1', F_2', \ldots, F_7'$ for all seven FS methods. This intersection ensures that only features selected as significant by each method are retained regardless of their specific rankings or values within each method list. As a result, the final set of selected features comprises those that not only surpass the threshold within each FS method but also appear in the intersection of all seven lists.

The proposed ensemble approach combines multiple filter-based FS methods ensuring that only the most robust and statistically significant features are retained. By aggregating results across different methods, this approach captures features that consistently demonstrate relevance, even under varying selection criteria. This process leads to a more thorough and effective dimensionality reduction, as it systematically eliminates redundant or irrelevant features, reducing noise in the dataset.

3.3.3 Model Training: Machine Learning Classifiers

The aim of this step is to evaluate the effectiveness of the proposed ensemble FS approach across a range of ML algorithms. In this step, the feature list created in Step II is used to train six distinct ML classifiers: RF, DT, KNN, NB, NN, and GB. These classifiers are widely adopted within research communities and allow to evaluate selected features through a diverse set of capabilities. Each classifier offers unique strengths: RF, an ensemble method, effectively handles high-dimensional data and provides insights into feature importance; DT is a straightforward, interpretable model that highlights how features contribute to class distinctions. KNN, as a distance-based classifier, assesses the distance nearer to the neighbor of features within classes, while NB, a probabilistic model, is computationally efficient and offers a baseline for feature relevance. NN captures complex, non-linear relationships, making it valuable for assessing intricate patterns, and GB, an ensemble method, enhances model accuracy by capturing feature interactions and complex patterns. This combination of both simpler and more complex models enables a thorough and balanced assessment, confirming the selected features' robustness, interpretability, and predictive power across multiple perspectives. Together, these classifiers validate that the selected features are relevant, stable, and capable of contributing to improved model accuracy in various learning scenarios. Each of the six ML classifiers is trained using the extracted feature set, which comprises the features identified through the ensemble FS approach. The use of a reduced feature set is intended to simplify the model training and reduce the consumption of computing resources, while still accurately classifying normal and attack samples.

3.3.4 Performance Evaluation Parameters

Following the training phase, the performance and resource utilization of the trained models are evaluated. Model performance is assessed using several criteria, including accuracy, TPR, TNR, FPR, and FNR, which provide a measure of the model's effectiveness. Training time (measured as wall-clock time), testing time, CPU, and memory utilization are monitored separately to evaluate the efficiency of the model during execution [160]. This approach ensures a comprehensive assessment of both the effectiveness and efficiency of the ML-based IDS models.

Accuracy: This term represents the percentage of correctly predicted network traffic or measures the correctness of a classification model. It is calculated by using the formula given in Equation (3.3).

$$\text{Accuracy } \% = \frac{TP + TN}{TP + TN + FP + FN} \times 100 \tag{3.3}$$

- True Positive (TP): The number of attack instances correctly identified as attacks.
- True Negative (TN): The number of normal instances correctly identified as normal.
- False Positive (FP): The number of normal instances incorrectly classified as attacks.
- False Negative (FN): The number of attack instances incorrectly classified as normal.

True Positive Rate (TPR): It is used to measure the proportion of ac-

tual positive instances that are correctly identified as positive by a classification or detection system, as given in Equation (3.4).

$$\text{TPR } \% = \frac{TP}{TP + FN} \times 100 \tag{3.4}$$

False Positive Rate (FPR): It is used to measure the proportion of actual negative instances that are incorrectly classified as positive by a detection system, as represented in Equation (3.5).

$$\text{FPR } \% = \frac{FP}{FP + TN} \times 100 \tag{3.5}$$

True Negative Rate (TNR): It is used to measures the proportion of actual negative instances that are correctly identified as negative by a detection system. TNR is calculated by using Equation (3.6).

$$\text{TNR } \% = \frac{TN}{TN + FP} \times 100 \tag{3.6}$$

False Negative Rate (FNR): It is used to measure the proportion of positive actual instances that are incorrectly classified as negative by a detection system, calculated using Equation (3.7).

$$\text{FNR } \% = \frac{FN}{FN + TP} \times 100 \tag{3.7}$$

Training Time: Training time encompasses the total duration, measured as wall-clock time, during which an ML algorithm performs dataset preprocessing, FS,

model training, and optimization to build a classification model. The training time is calculated using the formula provided in Equation (3.8):

$$\text{Training Time} = \text{End\_Time}_{\text{train}} - \text{Start\_Time}_{\text{train}} \tag{3.8}$$

where $\text{Start\_Time}_{\text{train}}$ is the time recorded just before the commencement of the training process and $\text{End\_Time}_{\text{train}}$ is the time recorded immediately after the training process ends.

Testing Time: Testing time is the time spent by the trained ML classifier to process an input instance and produce a prediction. This metric is essential for applications requiring real-time predictions. The testing time is calculated using Equation (3.9):

$$\text{Testing Time} = \text{End\_Time}_{\text{test}} - \text{Start\_Time}_{\text{test}} \tag{3.9}$$

where $\text{Start\_Time}_{\text{test}}$ is the time recorded just before the testing or prediction process begins and $\text{End\_ Time}_{\text{test}}$ is the time recorded immediately after the testing or prediction process ends. To measure the time duration of the training and testing phases of the classifier, we have imported the Python's time module, a specialized library designed to measure the code execution time in milliseconds. It should be noted that these time measurements can vary based on factors such as the hardware specifications, dataset complexity, and the specifics of the ML algorithm used.

Memory in Training Phase: During the training phase, memory utilization refers to the amount of computer memory actively used during tasks that often

involve dataset processing, FS process (in this study, the FS process is taken as an essential aspect during the training process), performing complex calculations, and storing intermediate results. The memory consumption during the training phase is calculated as given in Equation (3.10).

$$\text{Memory}_{\text{train}} = \text{Memory}_{a_{\text{train}}} - \text{Memory}_{b_{\text{train}}} \tag{3.10}$$

where $\text{Memory}_{a_{\text{train}}}$ is the memory usage measured before the training process starts. $\text{Memory}_{b_{\text{train}}}$ is the memory usage measured immediately after the training process ends.

Since the experiment is conducted in a controlled environment where only the training process affects memory usage, the formula estimates the memory consumption specifically for the training phase. Because no external processes or background tasks interfere with memory usage during the training phase, additional adjustments or supplementary memory measurement methods are not required. The controlled environment ensures that this calculation accurately reflects the memory demands of training-related tasks alone, making it a straightforward and valid approach to assessing memory utilization.

Memory in Testing Phase: In the testing phase, memory is utilized for processing input features, loading the trained model, and prediction results during the application of a trained model to test data samples is given in Equation (3.11).

$$\text{Memory}_{\text{test}} = \text{Memory}_{a_{\text{test}}} - \text{Memory}_{b_{\text{test}}} \tag{3.11}$$

where $\text{Memory}_{a_{\text{test}}}$ is the memory usage measured before the testing process starts.

$\text{Memory}_{b_{\text{test}}}$ is the memory usage measured immediately after the testing process ends.

CPU usage in Training: The training phase involves matrix calculations, weight updates, and FS, contributing to CPU utilization. To estimate CPU usage during training, the formula can be expressed as given in Equation (3.12):

$$\text{CPU}_{\text{train}} = \frac{\sum \text{CPU measurements}}{\text{Number of measurements}} \tag{3.12}$$

where *CPU measurements* represent individual measurements of CPU usage taken at regular intervals during the training phase. The *Number of measurements* is the total count of these individual measurements.

During the testing phase, the CPU processes input data through the trained model. The formula for calculating CPU usage during the testing phase of an ML model is similar to that used for the training phase. It involves measuring CPU usage at regular intervals during the testing process and then calculating the average of these measurements. This study employs the 'psutil' library [161], for the effective surveillance of resource consumption during both the training and testing phases of ML model development. The psutil retrieve information on resource utilization and process management. Memory usage is quantified in megabytes (MB) and CPU usage is represented in percentage (%).

3.4 Summary

In this chapter, a feature convergence method named ELIDS is introduced. ELIDS aims at identifying highly significant features by converging the results of various filter-based FS techniques. These techniques are utilized to identify a set of features, enabling the development of lightweight models for intrusion detection systems capable of identifying DoS/DDoS attacks. ELIDS converges the results of seven filter-based FS techniques, namely MI, Chi-squared, F-Score, PCC, VT, MAD, and DR. Lightweight models are developed using a set of six machine learning algorithms, namely RF, DT, KNN, NB, NN, and GB. These models can identify DoS/DDoS attacks by converging on the most significant features.

Owing to the fact that ELIDS follows statistical filter-based FS methods, information incompleteness can affect the actual data distribution, leading to inappropriate FS. In order to avoid this issue, the full data is used in the FS process to ensure that the most informative features are identified by ELIDS. The generalization abilities of the ML models trained with the identified features will be verified only in the cross-domain testing in the next chapter.

The upcoming chapter presents the experiment results investigating the efficacy of the proposed ELIDS. It analyses the performance of the proposed ELIDS scheme as a lightweight IDS mechanism in resource-limited IoT devices. Moreover, it analyses the efficiency of the proposed ELIDS scheme in the existing as well as in the unseen datasets.

CHAPTER 4

Performance Evaluation of ELIDS for ML-Enabled IDS

4.1 Introduction

This chapter discusses a comprehensive analysis of the experimental results using the proposed ELIDS. It encompasses the evaluation results conducted through both in-domain and cross-domain. As illustrated in Figure 4.1 The in-domain evaluation assesses the performance of ML classifiers using test samples drawn from the same dataset (i.e. TON_IoT) employed for training the ML classifiers. In contrast, cross-domain testing evaluates trained ML models performance using the test samples from a different dataset (i.e., BoT-IoT). The cross-domain testing helps to rigorously examine the model's adaptability for unseen data, particularly in real-world network traffic. Furthermore, the detailed analysis is presented across the six ML classifiers based on the evaluation parameters including accuracy, training and

testing times, confusion matrix parameters including FPR, FNR, TPR, and TNR, memory utilization, and CPU usage.

4.2 In-domain and Cross-domain Datasets

Datasets are fundamental components in designing and evaluating ML-based classification models that can be integrated within an IDS. The dataset provides the essential samples in acquiring the ability to distinguish between benign and attack traffic. An in-domain dataset consists of data from the same domain as the training set, sharing similar characteristics such as features, distribution, and context. This similarity allows the model to perform under familiar conditions with minimal variation between the training and testing phases. In contrast, a cross-domain dataset originates from a different domain than the training data and exhibits distinct characteristics, distributions, and contextual differences. Evaluating machine learning models with cross-domain datasets helps assess their ability to generalize and adapt to previously unseen environments—an essential aspect for real-world IDS applications.

This section elaborates the adopted datasets, namely TON_IoT and BoT-IoT, that are used for executing the ELIDS approach by shedding light on their origin and composition. Furthermore, we explore the relevance between the chosen datasets, a key factor essential for conducting meaningful cross-domain testing.

4.2.1 In-domain: TON_IoT Dataset

The TON_IoT dataset [29] is a benchmark resource for Industry 4.0/IoT and IIoT

cybersecurity research, generated from a large-scale testbed at UNSW Canberra's Cyber Range and IoT Labs. It incorporates heterogeneous data sources, including IoT/IIoT sensor telemetry, system logs from Windows and Linux environments, and network traffic traces, reflecting realistic IoT conditions. The dataset is publicly available in CSV and PCAP formats and contains both normal and malicious records. It covers nine types of attacks—DoS, DDoS, ransomware, password cracking, reconnaissance, MITM, backdoors, and web-based exploits such as XSS and SQL injection. Organized into raw, processed, and train-test directories, the dataset provides a large volume of labeled records suitable for robust experimentation. In this study, it is used for addressing the binary classification problem of DoS/DDoS detection.

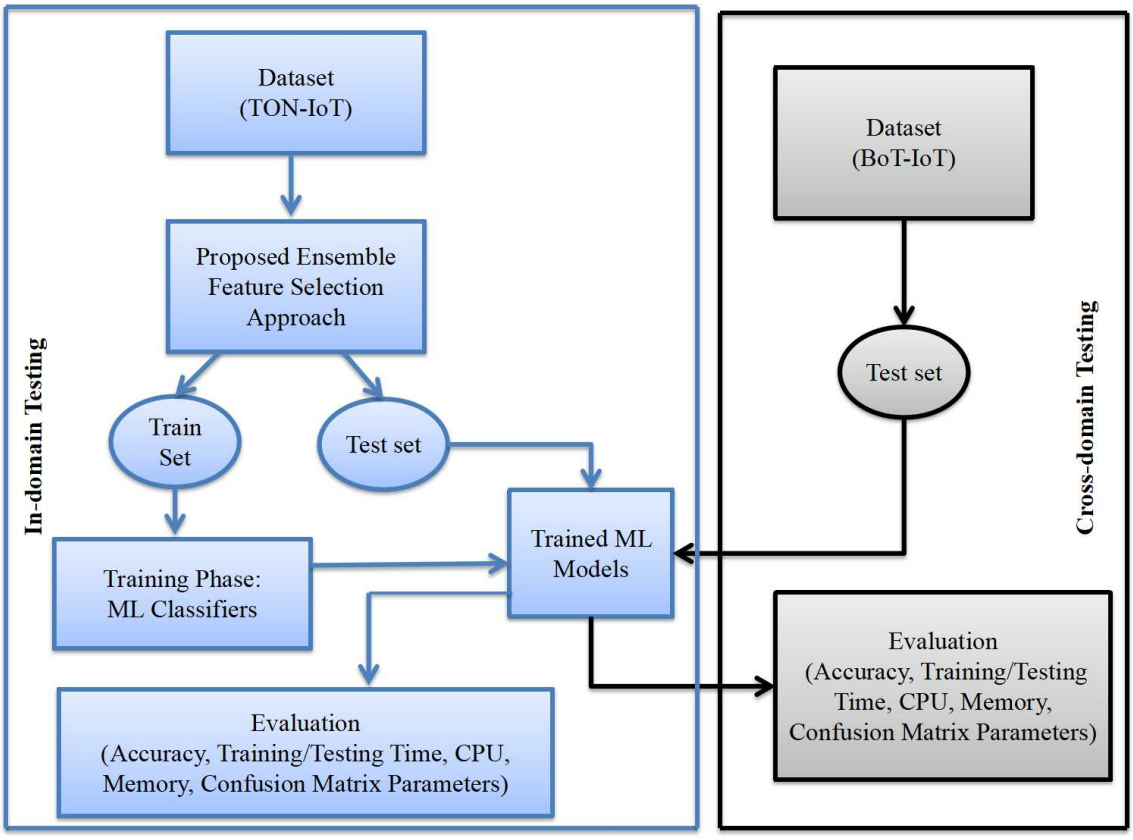


Figure 4.1: Flow diagram of In-domain and Cross-domain testing

The dataset comprises five main directories: train_test, description_stats, raw_datasets, processed, and security_Event_groundtruth datasets. In this chapter, the analysis is focused on the train_test_datasets directory, which is further subdivided into four subdirectories: train_test_iot_dataset, train_test_linux, train_test_network, and train_test_windows_dataset. The experimental work in this study utilizes the train_test_network dataset, which contains a total of 461,043 records, encompassing both normal and attack instances. Normal instances are 300,000, while each of the nine attack types comprises 20,000 records, except for MITM, which includes 1,043 samples. From these records, a subset of 300,000 normal and 40,000 DoS/DDoS instances is selected. This study specifically targets the detection of DoS/DDoS attacks among the multiple attack types present in the TON_IoT dataset.

4.2.2 Cross-domain: BoT-IoT Dataset

The Bot-IoT dataset [31] is a well-labeled collection of data that encompasses a range of attack scenarios, generated within a practical test environment. This dataset, available to the public in both PCAP and CSV formats, comprises approximately 72 million instances, each consisting of 37 distinct features. The experimental setup out of which the dataset is prepared employs four machines running the Kali Linux operating system to replicate various attacks such as DOS, DDoS (HTTP, TCP, UDP), port scanning, key-logging, and data exfiltration. To ensure the presence of IoT devices within the network, MQTT-enabled devices are introduced using the Node-RED middleware in a virtualized network configuration. Furthermore, the

dataset encompasses five distinct IoT scenarios: a weather station, a smart fridge, motion-activated lights, a remotely activated garage door, and a smart thermostat. These scenarios employ the MQTT protocol to generate data, hence representing a diverse range of IoT functionalities.

4.2.3 Relevancy of Datasets

Relevance between the two adopted datasets is essential for cross-domain evaluation since it ensures that the evaluation process reflects the model's ability to generalize to real-world scenarios. When the dataset domains are relevant, the evaluation provides meaningful insights into how well the model can adapt and perform in a different but related context. The selection of datasets for this study is guided by the relevancy criteria presented in Table 4.1. The feature sets used in both datasets (i.e., TON_IoT and BoT-IoT) are quite similar when it comes to classifying DoS/DDoS attacks and normal network traffic. Specifically, we employ TON_IoT dataset for in-domain evaluation and BoT-IoT dataset for cross-domain evaluation.

Table 4.1 also elaborates on the relevance of in-domain and cross-domain datasets. The first parameter indicates that both datasets are designed for an IoT environment. This is important because datasets generated from IoT devices have higher value, as training an IDS model on such data leads to better performance in real-world IoT environments. Network traffic samples indicate that both datasets contain network samples. Although the TON_IoT dataset contains nine attack records, we have extracted only the DoS/DDoS attack samples. This extraction ensures that the in-domain dataset is well aligned with the cross-domain dataset.

Moreover, the samples of benign and attack packets are gathered in a testbed environment, which helps the ML-based IDS model learn the actual parameters of real-world networks and thus perform well based on adequate training. Finally, the datasets are publicly available and can be downloaded in PCAP and CSV formats.

Table 4.1: Relevance between TON_IoT and BoT-IoT (✓) Yes (X) No

Parameters	**TON_IoT**	**BoT-IoT**
Designed for IoT	✓	✓
Network traffic samples	✓	✓
DoS/DDoS records	✓	✓
Testbed environment	✓	✓
Publicly available	✓	✓
Files in CSV and PCAP format	✓	✓

4.3 Experiments and Results

This section presents experiments validating the ELIDS approach concept and provides insights into the results obtained. The section comprises of four parts namely, Experimental Setup, Ensemble Features: Experiments and Analysis, In-domain Evaluations: Training and Testing Phases, and Cross-domain Evaluations: Testing Phase, as detailed in subsequent sub-sections.

4.3.1 Experimental Setup

In this work, all experiments are performed using Scikit learn v_0.24.2 libraries, Pandas v_1.3.4, and NumPy v_1.20.3 under the distribution of anaconda with Python

version 3.9.7. Configuration for the computer system utilized for executing the experiments includes a Core (TM) i5-6600 CPU @ 3.30GHz processor, 8 GB memory, where the system is running Windows_10 Pro OS. Furthermore, the Python "psutil" library is utilized to monitor resource usage, which includes the monitoring of CPU utilization, memory consumption, and training and testing time required with each learning algorithm. A summary of the tools and techniques utilized for evaluating the proposed ELIDS is presented in Table 4.2.

Table 4.2: Summary of Experimental Setup

Resources	**Description**
CPU	Core (TM) i5-6600 @ 3.30GHz
Memory	8GB
Operating System	Windows 10 Pro
Development Environment	Anaconda (Jupyter Notebook)
Libraries	Scikit-learn v_0.24.2, Pandas v_1.3.4, NumPy v_1.20.3, time module and psutil

In the data preprocessing stage, Normal and DoS/DDoS samples are extracted, initially comprising 300,000 'Normal' and 40,000 'DoS/DDoS' records. To address class imbalance, an undersampling technique is applied to reduce the majority of class samples, resulting in a balanced dataset. This approach enhances training efficiency, prevents overfitting, and ensures fair representation of both classes. After preprocessing, the dataset consists of 80,000 samples, evenly distributed with 40,000 attack records and 40,000 normal records, maintaining a balanced class ratio for op-

timal model performance.

To further mitigate model bias, the dataset is partitioned into training and testing subsets using 10-fold cross-validation [162]. This approach improves randomness in sample selection, ensuring that the model is trained and evaluated on unbiased data across each fold. The 80,000 samples are divided into 10 distinct subgroups. Among these, 72,000 samples are assigned for training, while 8,000 samples are used for testing, as presented in Table 4.3. In each iteration, one fold (8,000 samples) is used for testing, while the remaining nine folds (72,000 samples) are initially used for ensemble feature selection (FS) and subsequently for model training. The reported results represent the average performance across all folds, thereby reducing the impact of any imbalance in individual partitions and providing a more generalized evaluation of the proposed method. Finally, the trained ML model undergoes validation through cross-domain evaluation using the BoT-IoT dataset.

Table 4.3: Dataset Distribution of Training and Testing Samples

Dataset	Total Instances	Normal Instances	Attack Instances
TON_IoT	80,000	40,000	40,000
Training Set (90%)	72,000	36,000	36,000
Testing Set (10%)	8,000	4,000	4,000

4.3.2 Phase I: Ensemble Features Experiments and Analysis

The adopted TON_IoT dataset consists of 43 features for building binary classification models by excluding the 'type' and 'label' features from the original dataset

of 45 features. These 43 features are ranked based on their significance by utilizing seven separate filter-based FS methods, hence generating seven distinct lists of ranked features.

A detailed representation of the ranked features using the considered FS methods is presented in Table 4.4. Based on the obtained results, it is pertinent to highlight that a major difference in feature ranking is observed by the seven FS methods where a feature ranked as one of the most significant by a given FS method is observed to be ranked as one of the least significant features by another FS method. As a proof of concept, the feature 'src_ip' which designates the attacker source IP, is ranked as '1' hence being the most significant feature in the dataset as per the MI FS method. Conversely, the same feature is ranked as '40', i.e. being one of the least significant features when evaluated by the χ^2 FS method. Having said that, it is also observed that 'ts', which designates the timestamp, consistently maintains its position as the most significant feature across all seven feature selection techniques, as highlighted in Table 4.4.

To address such a deviation in features' rankings exhibited by the individual filter-based FS methods, the proposed ELIDS approach selects a set of features by applying its two-fold filtration process that converges the selection process towards the most significant features.

Table 4.4: Ranking and corresponding values of the features set

Feature Rank	IG	Value	chi2	Value	F-score	Value	R	Value	VT	Value	MAD	Value	DR	Value
1	src_ip	0.6981	dst_port	6194.615	ts	29739.59	ts	0.5206	ts	0.1437	ts	0.3061	ts	1.000932
2	ts	0.6932	ts	4532.246	dst_port	17960.16	dns_rcode	0.1895	src_port	0.1176	src_port	0.3051	weird_notice	1.000646
3	dst_ip_bytes	0.5666	service	1916.262	proto	11711.54	src_bytes	0.0257	dns_RD	0.1171	conn_state	0.3043	conn_state	1.000201
4	src_ip_bytes	0.4908	src_port	1789.606	src_port	8084.196	conn_state	0.0242	conn_state	0.1155	dns_RD	0.269	dst_port	1.000164
5	conn_state	0.4387	dns_rcode	1634.78	dst_ip	6202.475	dst_port	0.0099	dns_query	0.0916	service	0.2406	ssl_resumed	1.000156
6	dst_port	0.433	dns_qtype	1234.88	service	5130.241	proto	0.0095	dns_rejected	0.0847	proto	0.2325	http_version	1.00013
7	duration	0.397	dns_AA	1118.561	dns_rcode	2978.931	src_ip	0.0095	service	0.0705	dns_query	0.2285	src_bytes	1.000106
8	dst_ip	0.3687	proto	931.8127	dns_AA	2436.222	service	0.0041	proto	0.0614	dns_rejected	0.2265	ssl_version	1.000105
9	src_port	0.3425	dst_ip	495.4338	dns_qtype	2391.88	dst_ip	0.0035	dns_RA	0.0607	dns_RA	0.1964	weird_name	1.0001
10	dst_pkts	0.2703	dns_query	444.8161	dns_RA	920.4354	src_port	0.0014	dns_AA	0.0598	dns_AA	0.1952	dst_ip	1.000067
11	src_pkts	0.1867	dns_RA	434.64	dns_query	716.4414	duration	0	dst_port	0.0343	dst_port	0.116	http_resp_mime	1.000061
12	dst_bytes	0.1677	dns_qclass	243.7343	conn_state	407.6438	http_resp_mime	-0.0007	dst_ip	0.0282	dst_ip	0.1012	ssl_cipher	1.000055
13	dns_query	0.1504	weird_notice	157.213	dns_qclass	245.7706	http_user_agent	-0.0021	dns_rcode	0.0257	dns_rcode	0.0837	http_user_agent	1.000054
14	src_bytes	0.1429	conn_state	104.3945	weird_notice	157.8528	src_pkts	-0.0035	dns_qtype	0.0093	dns_qtype	0.0307	src_port	1.000054
15	dns_rejected	0.0984	weird_addl	67	weird_addl	96.653	http_resp_body	-0.0036	src_ip	0.0084	src_ip	0.0229	http_method	1.000034
16	proto	0.0931	ssl_resumed	54.069	ssl_resumed	88.5385	http_status_code	-0.0037	dns_qclass	0.003	dns_qclass	0.0061	dst_bytes	1.000023
17	dns_RD	0.0782	ssl_established	51.7689	ssl_established	81.7733	dst_pkts	-0.004	weird_notice	0.0021	weird_notice	0.0042	src_ip_bytes	1.000012
18	dns_qtype	0.0687	http_trans_dep	36.4033	weird_name	70.6895	src_ip_bytes	-0.0049	weird_addl	0.0006	weird_addl	0.0017	dst_pkts	1.000012
19	service	0.033	weird_name	34.5145	duration	53.071	http_req_body	-0.0055	ssl_established	0.0005	ssl_established	0.0015	ssl_issuer	1.000009
20	dns_rcode	0.0251	ssl_version	29.3889	dst_bytes	46.7505	http_orig_mime	-0.0059	ssl_resumed	0.0004	ssl_resumed	0.0014	proto	1.000009
21	dns_AA	0.0247	dns_RD	23.9831	ssl_version	36.5166	http_uri	-0.0066	http_version	0.0004	weird_name	0.0011	ssl_subject	1.000008
22	dns_RA	0.0171	ssl_cipher	14.8923	dns_RD	35.6908	http_trans_dep	-0.0067	ssl_version	0.0003	src_bytes	0.0009	service	1.000008

23	missed_bytes	0.0106	dst_bytes	14.5835	ssl_cipher	24.8384	http_method	-0.0072	src_bytes	0.0003	http_version	0.0008	http_req_body	1.000008
24	dns_qclass	0.0096	http_version	7.5294	missed_bytes	7.7776	http_version	-0.0097	weird_name	0.0003	ssl_version	0.0008	missed_bytes	1.000007
25	ssl_issuer	0.0025	dns_rejected	4.2008	http_version	7.5331	ssl_cipher	-0.0176	http_status_code	0.0002	http_status_code	0.0006	src_pkts	1.000005
26	weird_name	0.0015	missed_bytes	3.532	dns_rejected	7.2776	dns_RD	-0.0211	http_resp_mime	0.0002	ssl_cipher	0.0005	duration	1.000005
27	ssl_version	0.0012	http_orig_mime	2.5	src_ip	7.2567	ssl_version	-0.0214	http_user_agent	0.0002	http_uri	0.0005	dns_qclass	0
28	ssl_established	0.0011	http_req_body	2.2415	http_method	4.1901	weird_name	-0.0297	ssl_cipher	0.0002	http_resp_mime	0.0005	dst_ip_bytes	0
29	http_version	0.0008	http_uri	2.2141	http_trans_dep	3.6374	ssl_established	-0.032	http_uri	0.0002	http_user_agent	0.0005	dns_qclass	0
30	http_resp_mime	0.0007	http_method	2.1407	http_uri	3.4922	ssl_resumed	-0.0332	http_method	0.0001	http_method	0.0004	http_uri	0
31	weird_notice	0.0006	ssl_subject	1.1905	http_orig_mime	2.7779	weird_addl	-0.0347	dst_bytes	0.0001	dst_bytes	0.0004	src_ip	0
32	ssl_cipher	0.0002	src_bytes	1.0417	http_req_body	2.4411	weird_notice	-0.0444	src_ip_bytes	0	duration	0.0002	http_trans_dep	0
33	ssl_subject	0.0001	http_resp_body	1.0047	src_ip_bytes	1.9469	dns_qclass	-0.0553	dst_pkts	0	src_ip_bytes	0.0002	http_orig_mime	0
34	http_method	0	ssl_issuer	0.9	ssl_subject	1.3158	dst_bytes	-0.0712	ssl_issuer	0	http_trans_dep	0.0001	ssl_established	0
35	http_uri	0	src_ip_bytes	0.8299	dst_pkts	1.2538	dns_query	-0.0942	http_orig_mime	0	dst_pkts	0.0001	http_status_code	0
36	http_req_body	0	http_status_code	0.7871	http_status_code	1.0839	dns_RA	-0.1067	ssl_subject	0	missed_bytes	0.0001	dns_rejected	0
37	http_resp_body	0	dst_pkts	0.5252	http_resp_body	1.012	dns_qtype	-0.1704	http_req_body	0	src_pkts	0.0001	dns_RD	0
38	http_status_code	0	src_pkts	0.3159	ssl_issuer	1	dns_AA	-0.1719	http_trans_dep	0	ssl_issuer	0.0001	dns_rcode	0
39	http_user_agent	0	http_user_agent	0.2381	src_pkts	0.9713	ssl_subject	-0.2455	missed_bytes	0	http_orig_mime	0.0001	dns_RA	0
40	http_orig_mime	0	src_ip	0.1528	http_user_agent	0.3464	ssl_issuer	-0.2682	duration	0	ssl_subject	0.0001	dns_query	0
41	ssl_resumed	0	dst_ip_bytes	0.0738	dst_ip_bytes	0.148	dst_ip_bytes	-0.303	src_pkts	0	http_req_body	0.0001	dns_qtype	0
42	weird_addl	0	http_resp_mime	0.0273	http_resp_mime	0.0342	dns_rejected	-0.3574	dst_ip_bytes	0	dst_ip_bytes	0	dns_AA	0
43	http_trans_dep	0	duration	0.0196	src_bytes	0.0001	missed_bytes	-0.4282	http_resp_body	0	http_resp_body	0	weird_addl	0

In the first fold, ELIDS reduces the complete features set by filtering those features ranked below the median value (i.e 22) calculated for the set of features presented in Table 4.4, where for the case of TON_IoT dataset, features having a rank below 22 are filtered out. This is performed over all the seven lists generated by the seven filter-based FS techniques. As a result, the set of features having a rank greater than or equal to 22 is kept while filtering the remaining features, as presented in Table 4.5. In the second fold, only the features that are present in all seven filtered lists (generated in the first fold) are kept while filtering out all other features. The set of features that are present in each of the seven features lists are highlighted in Table 4.5 for the sake of clarity. In this work, it is observed that a total of seven features only are found to be intersecting in the generated lists, where a detailed description of the finally generated set of features is provided in Table 4.6.

The seven features represented by Table 4.6 exhibit a set in which each feature is speculated to have a high probability of impact in the effective binary classification of DDoS attacks from normal traffic. However, a feature satisfying the proposed ELIDS criterion in six of the considered FS techniques, but failing to do so in the seventh FS technique would be discarded as it would cast doubts in its significance for having an effective impact on the classification model built while incorporating that particular feature. In Table 4.4, it is observed that 'src_ip' satisfies the criterion set by the ELIDS approach by ranking it as '1', '7', '15', '15' by MI, PCC, VT and MAD, respectively. However, this feature fails to satisfy the same criterion by the remaining three FS techniques, hence deeming it as not suitable for building the classification model.

Table 4.5: List of Features and Corresponding Values Below Median

Feature Rank	IG	Value	χ^2	Value	F-score	Value	R	Value	VT	Value	MAD	Value	DR	Value
1	src_ip	0.6981	dst_port	6194.6	ts	29739.6	ts	0.5206	ts	0.1437	ts	0.3061	ts	1.000932
2	ts	0.6932	ts	4532.2	dst_port	17960.2	dns_rcode	0.1895	src_port	0.1176	src_port	0.3051	weird_notice	1.000646
3	dst_ip_bytes	0.5666	service	1916.3	proto	11711.5	src_bytes	0.0257	dns_RD	0.1171	conn_state	0.3043	conn_state	1.000201
4	src_ip_bytes	0.4908	src_port	1789.6	src_port	8084.2	conn_state	0.0242	conn_state	0.1155	dns_RD	0.269	dst_port	1.000164
5	conn_state	0.4387	dns_rcode	1634.8	dst_ip	6202.5	dst_port	0.0099	dns_query	0.0916	service	0.2406	ssl_resumed	1.000156
6	dst_port	0.433	dns_qtype	1234.9	service	5130.2	proto	0.0095	dns_rejected	0.0847	proto	0.2325	http_version	1.00013
7	duration	0.397	dns_AA	1118.6	dns_rcode	2978.9	src_ip	0.0095	service	0.0705	dns_query	0.2285	src_bytes	1.000106
8	dst_ip	0.3687	proto	931.81	dns_AA	2436.2	service	0.0041	proto	0.0614	dns_rejected	0.2265	ssl_version	1.000105
9	src_port	0.3425	dst_ip	495.43	dns_qtype	2391.9	dst_ip	0.0035	dns_RA	0.0607	dns_RA	0.1964	weird_name	1.0001
10	dst_pkts	0.2703	dns_query	444.82	dns_RA	920.4	src_port	0.0014	dns_AA	0.0598	dns_AA	0.1952	dst_ip	1.000067
11	src_pkts	0.1867	dns_RA	434.64	dns_query	716.4	duration	0	dst_port	0.0343	dst_port	0.116	http_resp_mime	1.000061
12	dst_bytes	0.1677	dns_qclass	243.73	conn_state	407.6	http_resp_mime	-0.0007	dst_ip	0.0282	dst_ip	0.1012	ssl_cipher	1.000055
13	dns_query	0.1504	weird_notice	157.21	dns_qclass	245.8	http_user_agent	-0.0021	dns_rcode	0.0257	dns_rcode	0.0837	http_user_agent	1.000054
14	src_bytes	0.1429	conn_state	104.39	weird_notice	157.9	src_pkts	-0.0035	dns_qtype	0.0093	dns_qtype	0.0307	src_port	1.000054
15	dns_rejected	0.0984	weird_addl	67	weird_addl	96.7	http_resp_body	-0.0036	src_ip	0.0084	src_ip	0.0229	http_method	1.000034
16	proto	0.0931	ssl_resumed	54.069	ssl_resumed	88.5	http_status_code	-0.0037	dns_qclass	0.003	dns_qclass	0.0061	dst_bytes	1.000023
17	dns_RD	0.0782	ssl_established	51.769	ssl_established	81.8	dst_pkts	-0.004	weird_notice	0.0021	weird_notice	0.0042	src_ip_bytes	1.000012
18	dns_qtype	0.0687	http_trans_dep	36.403	weird_name	70.7	src_ip_bytes	-0.0049	weird_addl	0.0006	weird_addl	0.0017	dst_pkts	1.000012
19	service	0.033	weird_name	34.515	duration	53.1	http_req_body	-0.0055	ssl_established	0.0005	ssl_established	0.0015	ssl_issuer	1.000009
20	dns_rcode	0.0251	ssl_version	29.389	dst_bytes	46.8	http_orig_mime	-0.0059	ssl_resumed	0.0004	ssl_resumed	0.0014	proto	1.000009
21	dns_AA	0.0247	dns_RD	23.983	ssl_version	36.5	http_uri	-0.0066	http_version	0.0004	weird_name	0.0011	ssl_subject	1.000008
22	dns_RA	0.0171	ssl_cipher	14.892	dns_RD	35.7	http_trans_dep	-0.0067	ssl_version	0.0003	src_bytes	0.0009	service	1.000008

Table 4.6: Features selected with Ensemble-based FS approach

Feature	Description
ts	A timestamp, within the TON_IoT dataset, is a data field or column that signifies the date and time of a specific event or observation. It contains numerical values representing the timestamp associated with sensor reading data.
src_port	The port number employed by the transmitting device serves as a unique identifier for its end of the communication. It also helps the receiving device in determining where to direct the response.
dst_ip	Each device connected to a network is assigned a unique IP address, which serves as its identifier. The TON_IoT dataset incorporates the "dst_ip" feature, which is specific to hosts.
dst_port	In TON_IoT dataset, dst_port number is used by the receiving device to identify which service or application should handle the incoming data.
proto	The field within the TON_IoT dataset signifies the transport layer protocol of the flow connection, commonly encompassing values such as 'tcp', 'udp', and 'icmp'.
service	In TON_IoT dataset, the service field represents the specific network service or protocol associated with a communication, encompassing entries such as 'dns', 'ssl', 'dhcp', 'http', 'ftp', 'dce_rpc', 'gssapi', 'smb;gssapi', and 'smb'.
conn_state	This field in the dataset contains a code detailing the state of the connection in the network flow. Various connection states, such as S0 (connection without reply), S1 (connection established), and REJ (connection attempt rejected), can be identified in this feature.

Through this strict criterion, the proposed ELIDS approach not only aims to select the most significant features indicated by all considered FS techniques but also attempts to reduce feature dimensionality. This in turn limits the number of features involved in building the classification model and hence reduces resource consumption by the built classification model, as a major requirement in resource constraint IoT devices.

4.3.3 Phase II: In-domain Evaluations for Training and Testing

In-domain evaluations assess the classification model's performance using data from the domain it was trained on, focusing on the specific characteristics the model learned during training.

The features selected through the ensemble approach presented in Table 4.7 are subsequently employed to train ML models. These parameter configurations have been chosen based on their widespread adoption within the research community [163], [164], as they have been proven to produce reliable and accurate results in various studies. Additionally, the selection of these parameters takes into account the computational constraints of IoT devices, ensuring that the models remain efficient and feasible for real-world deployment without compromising performance.

Afterward, a subset of the preprocessed testing samples is utilized to further scrutinize the performance of the trained ML models. The evaluation rigorously examines the effectiveness of our proposed ELIDS approach, particularly in terms of accuracy and resource utilization. Moreover, this section comprehensively evalu-

ates the effectiveness of the proposed ELIDS approach in contrast to models trained using individual filter-based FS including MI, χ^2, F-score, PCC, VT, MAD, and DR techniques. It is shown in section 4.3.2 that ELIDS converges on 7 features, which are compared with the top 7 features from each method. Having said that, a previous study in [38] indicates that several ML algorithms perform optimally with precisely seven features.

For each conventional FS method and the proposed ELIDS approach, a total of six ML classifiers are developed (i.e. RF, DT, GB, KNN, NB, and NN). The parameters configuration for each ML classifier, used in this study is given in Table 4.7. For RF and DT, the settings are employed with Gini impurity as the splitting criterion, and the model is trained with the maximum depth 5. The NN is configured with five hidden layers, where the number of neurons decreases progressively from 7 neurons in the first hidden layer to just 2 neuron in the fifth hidden layer, using the ReLU activation function and the Adam solver. For GB, a learning rate of 0.1 and a maximum depth of 5 are used, with 100 estimators to balance model performance and training time. In the case of KNN, the number of neighbors is set to 268, based on the rule of thumb, the square root of 72,000 samples, ensuring the model strikes a balance between overfitting and underfitting [165]. In the KNN, setting algorithm = 'auto' allows scikit-learn to automatically select the most appropriate search method based on the dataset. Depending on the data distribution and dimensionality, the classifier chooses between Ball Tree, KD Tree, or brute-force search, ensuring efficient neighbor searches without requiring manual specification. Finally, the NB classifier uses a GaussianNB model, suitable

for continuous features. Each classifier is configured to optimize performance while accounting for the computational resources available and the specific characteristics of the dataset.

Table 4.7: Classifiers and their configuration

Classifier	Configuration
RF	RandomForestClassifier(n_estimators=100, max_depth=5 criterion='gini', min_samples_split=5, warm_start=True)
DT	DecisionTreeClassifier(criterion='gini', max_depth=5 min_samples_split=5)
GB	GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=5, warm_start=True)
KNN	KNeighborsClassifier(n_neighbors=268, algorithm='auto', weights='uniform')
NB	GaussianNB()
NN	MLPClassifier(hidden_layer_sizes=(7, 5, 4, 3, 2), activation='relu', solver='adam', max_iter=200)

The evaluation encompasses a total of 42 experiments, comprising seven filter-based FS methods combined with six ML classifiers. Furthermore, six experiments were conducted to train the same ML classifiers using the proposed ELIDS approach, hence aggregating total of 48 experiments. Figures 4.2 to 4.10 display the results of in-domain evaluations, showing accuracy, performance metrics, training and testing time, CPU usage, and memory consumption. The reported results represent the average across all 10 folds, ensuring robustness and avoiding bias from any single fold.

Figure 4.2 demonstrates that the six ML classifiers built using ELIDS consistently provide the most promising accuracy performance when compared to those classifiers built using the individual FS methods. Remarkably, ELIDS classifiers attain 99.83% accuracy when considering RF, DT, KNN, and GB algorithms, while also maintains accuracies exceeding 89% for NB and NN classifiers. However, clas-

sifiers built using the conventional FS techniques also exhibit 99.8% accuracy rates when considering RF, DT, and GB algorithms, but generally tend to fall in accuracy when considering KNN, NB, and NN algorithms. Overall, the conventional FS techniques along with the proposed ELIDS show the highest and similar accuracy performance over RF, DT, and GB classification models.

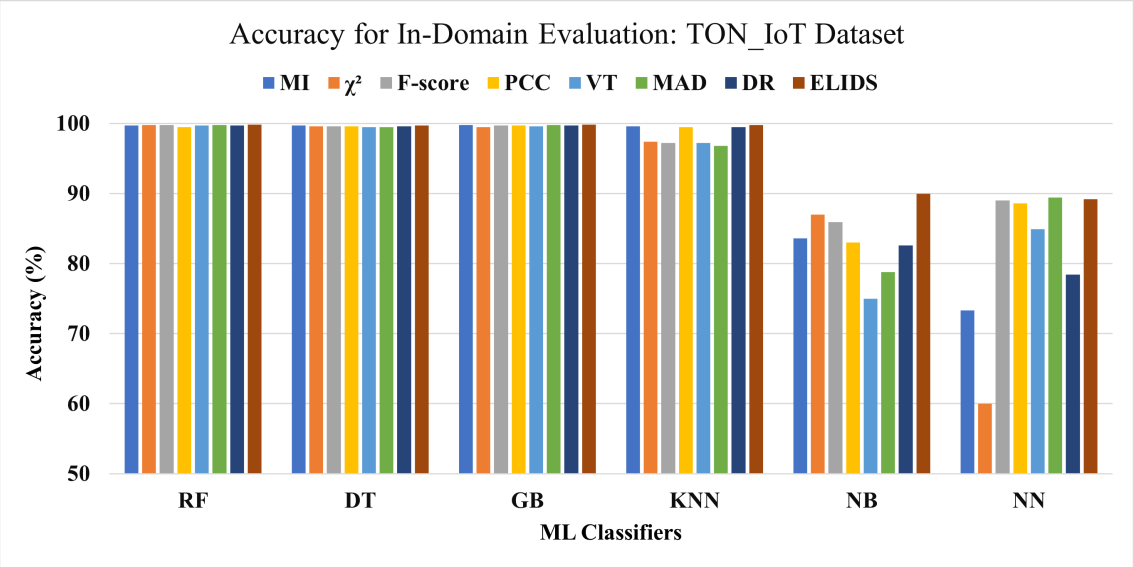


Figure 4.2: Performance evaluation of ML classifiers with ELIDS and filter-based FS methods

Performance measures are presented in Figures 4.3 (a, b, c, and d), for TPR, TNR, FNR, and FPR, respectively. As depicted by Figure 3(a), ELIDS based classifiers achieve TPR values more than 0.94 for all six ML algorithms, hence it signifies that almost these models correctly identify all attack cases. On the other hand, values obtained for FNR are depicted by Figure 4.3(c), which is in complement to those obtained in TPR. The FNR values of 0 indicate that no DDoS attack was classified as normal, except for the NN, which has a value of 0.06. Classifiers built based on the individual FS techniques tend to show similar TRP and FNR

performance as obtained by ELIDS when considering RF, DT, and GB algorithms. However, the counterpart classifiers built using KNN, NB, and NN exhibit a general fall in performance for both TPR and FNR. Similarly, ELIDS based classifiers obtain a TNR value of 1 for RF, DT, GB, and KNN, while achieving a TNR of 0.8 for NB and NN, as depicted by Figure 4.3(b).

Classifiers based on the conventional FS techniques obtained similar TNR performance to that of ELIDS when considering RF, DT, and GB algorithms. On the other hand, classifiers based on ELIDS and the individual FS techniques show an FPR of 0.01 value for the case of RF, DT, and GB algorithms. Results generally reveal that ELIDS based classifiers not only can correctly identify DDoS attacks but also generate low false alarms when compared with other individual FS methods.

Figure 4.4 shows the training times of six ML classifiers using ELIDS compared to individual FS methods. As given in Equation (3.8), training time encompasses data preprocessing, FS, model training, and optimization. MI-based classifiers have the longest training times, since MI is computationally intensive and the features it selects often require more extensive model optimization to achieve the best performance. In contrast, ELIDS combines MI with six FS methods and includes an ensemble fusion step that selects a small, optimized subset of features. This step effectively reduces redundancy and dimensionality, and because it is performed only once during preprocessing, the subsequent model training and optimization operate on the final reduced set of seven features. As a result, ELIDS-based classifiers avoid heavy computational overhead, achieving reasonable training times, for example, 168 ms for DT and 284 ms for RF.

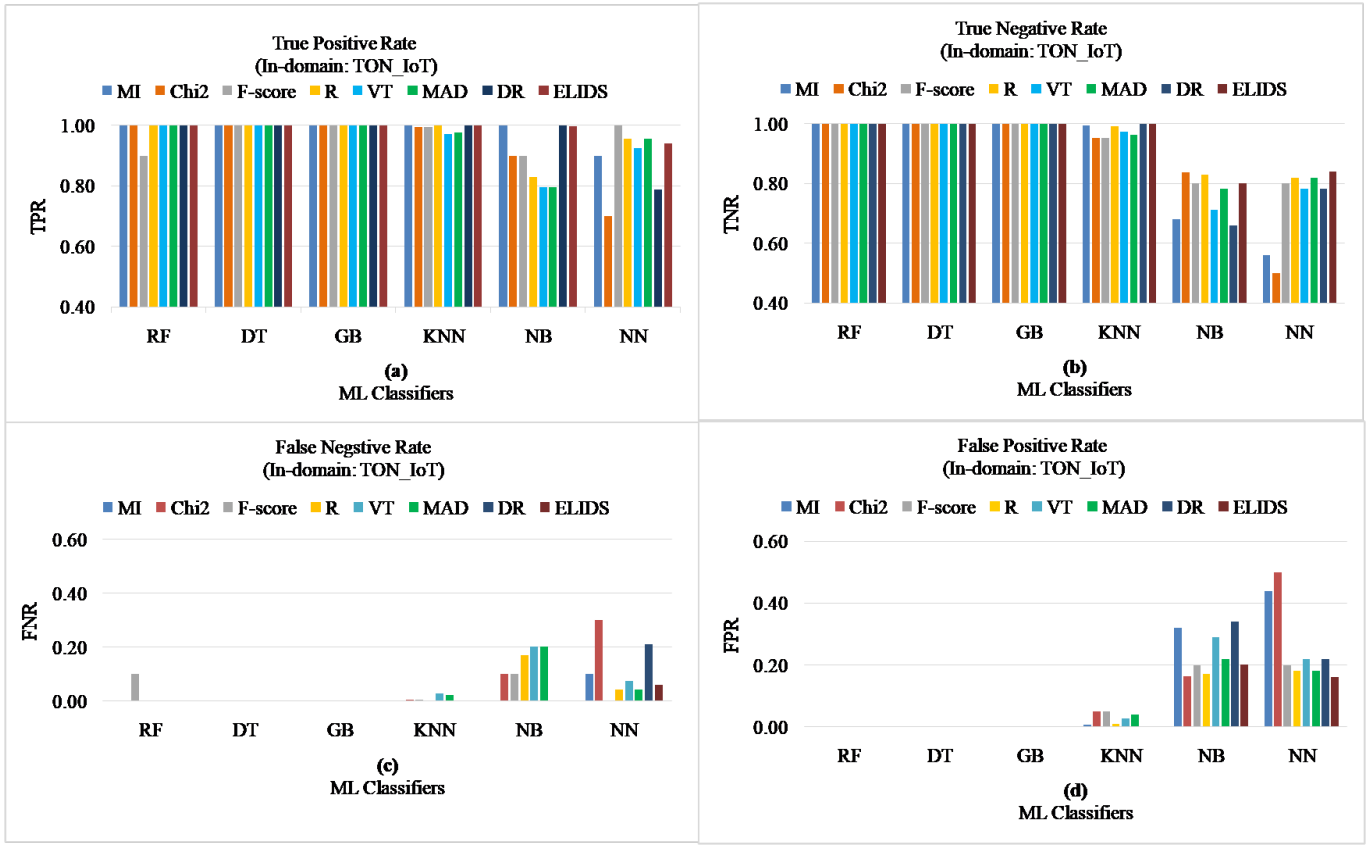
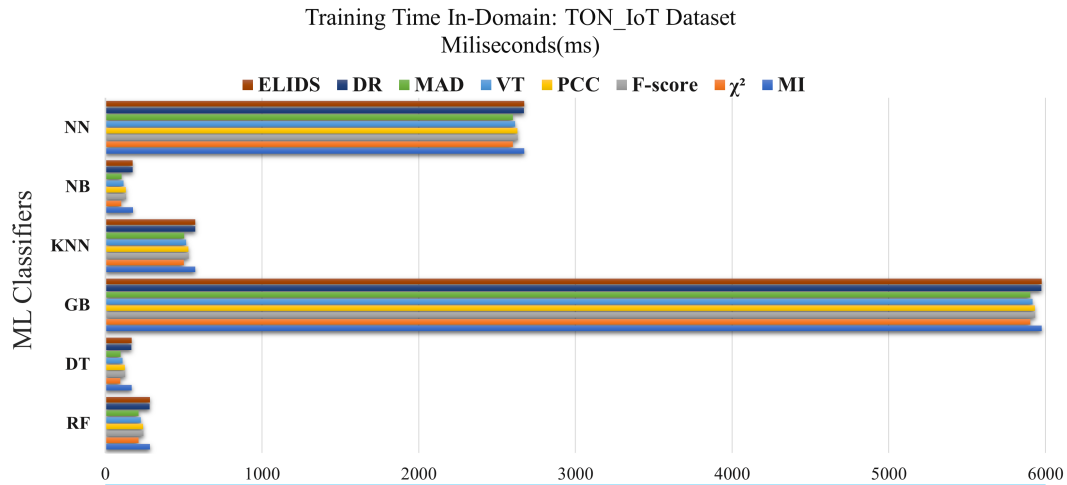


Figure 4.3: Performance metrics (a) TPR (b) TNR (c) FNR and (d) FPR

FS process is typically a part of preprocessing and once the features are selected, they are applied during the training and testing phase. In this study, the trained ML models have the same number of features (i.e., 7 features) as discussed earlier in this section, and the testing process primarily entails utilizing the trained model to make predictions, known as model inference. Therefore, any differences in testing time within ML algorithm using any of the studied FS methods are less significant.



	RF	DT	GB	KNN	NB	NN
ELIDS	284	168	5975	574	175	2673
DR	283	167	5974	573	174	2672
MAD	213	97	5904	503	104	2602
VT	226	110	5917	516	117	2615
PCC	240	124	5931	530	131	2629
F-score	242	126	5933	532	133	2631
χ^2	211	95	5902	501	102	2600
MI	285	169	5976	575	176	2674

Figure 4.4: Time required to train ML classifiers using ELIDS vs individual filter-based FS

The graph depicted in Figure 4.5 presents the testing time required to make predictions on 8,000 TON_IoT data samples while considering the six ML classification models. DT outperforms other classifiers with the lowest testing time of 0.93

msec, whereas KNN consumes the highest testing time of 856.8 msec. In addition, other classification models including RF (5.04 msec), GB (30.24 msec) and NB (1.68 msec) are also observed to require low testing time and hence maintain low process execution overheads over IoT end devices.

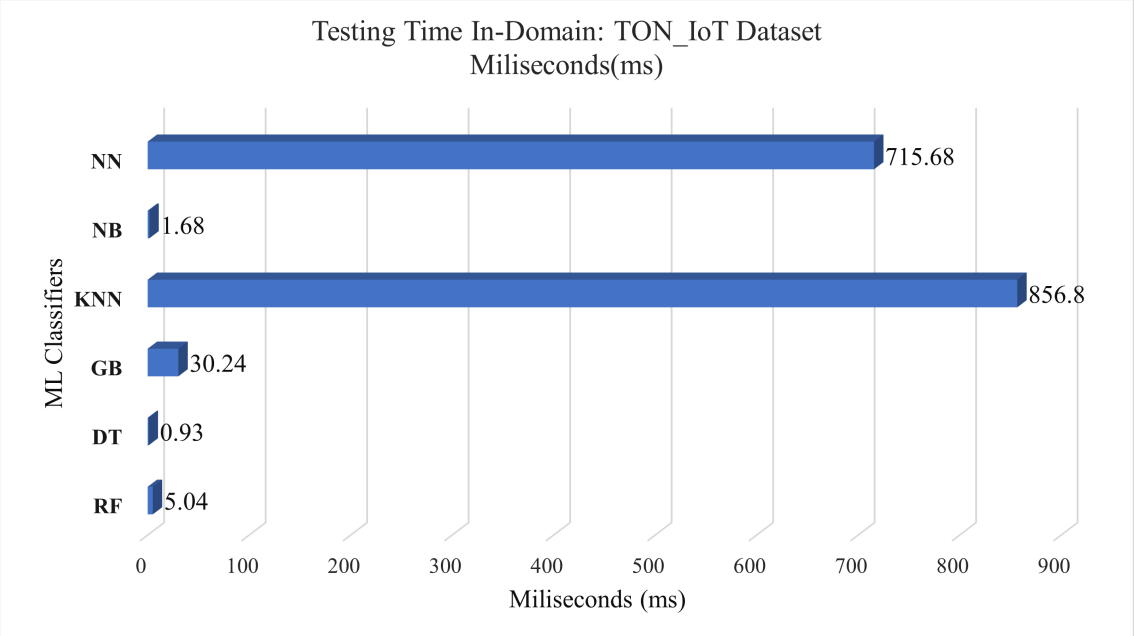
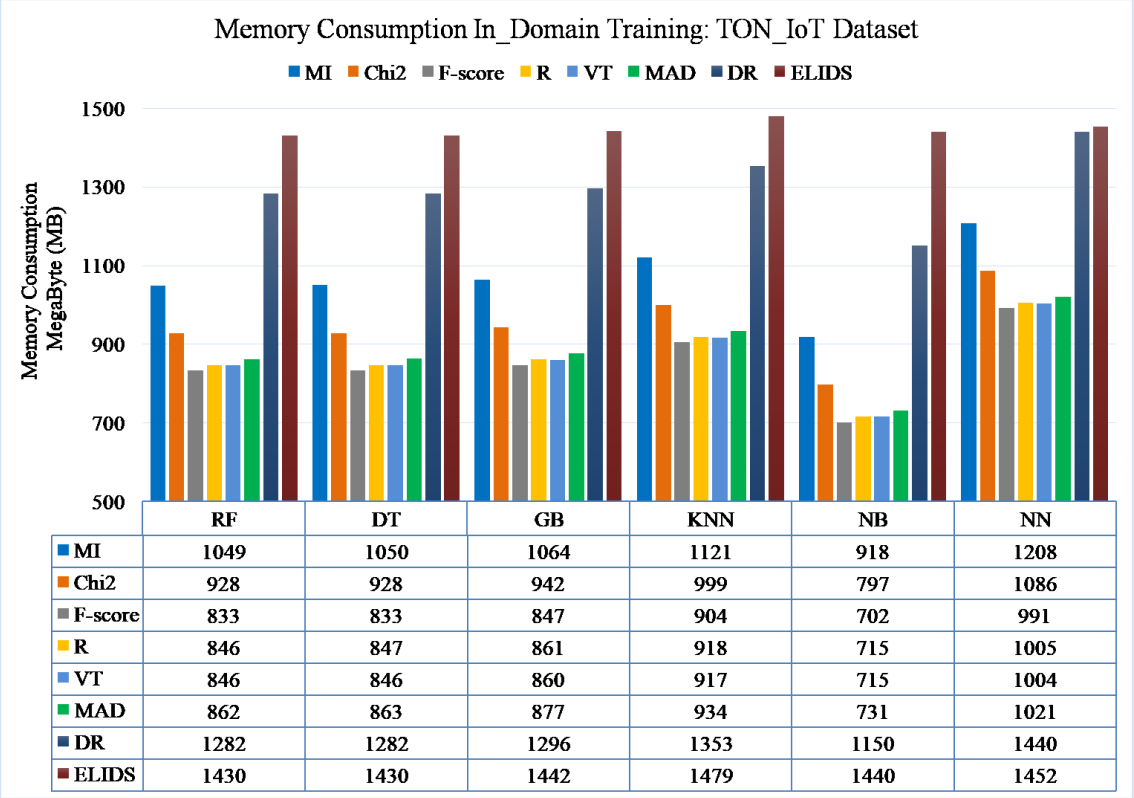


Figure 4.5: Testing time required by ML classifier

Figure 4.6 shows the memory consumption during the training phase for the six considered ML classifiers built out of the proposed ELIDS and the conventional FS techniques. ELIDS exhibits the highest memory consumption of all considered six ML classifiers as compared to the individual FS techniques, which is due to the ensemble approach employed in the FS process. Nevertheless, the average consumption indicates that the difference is not substantial, as ELIDS based classifiers consume an average of 1,445 MB of memory, while the ML classifiers trained using the features selected by the DR and MI methods consume an average of 1,301 MB and 1,068 MB of memory, respectively. Furthermore, it is noteworthy that NN and

KNN based classifiers exhibit on average the highest memory consumption across all types of FS techniques, which is quite evident given their resource-intensive nature. Having said that, the training process generally takes place on a resourceful system, such as a server or a cloud, as opposed to resource constraint IoT devices. Hence, the higher consumption of memory by ELIDS enabled classifiers becomes a less concerning issue due to the abundant amount of memory capacity over resourceful devices.

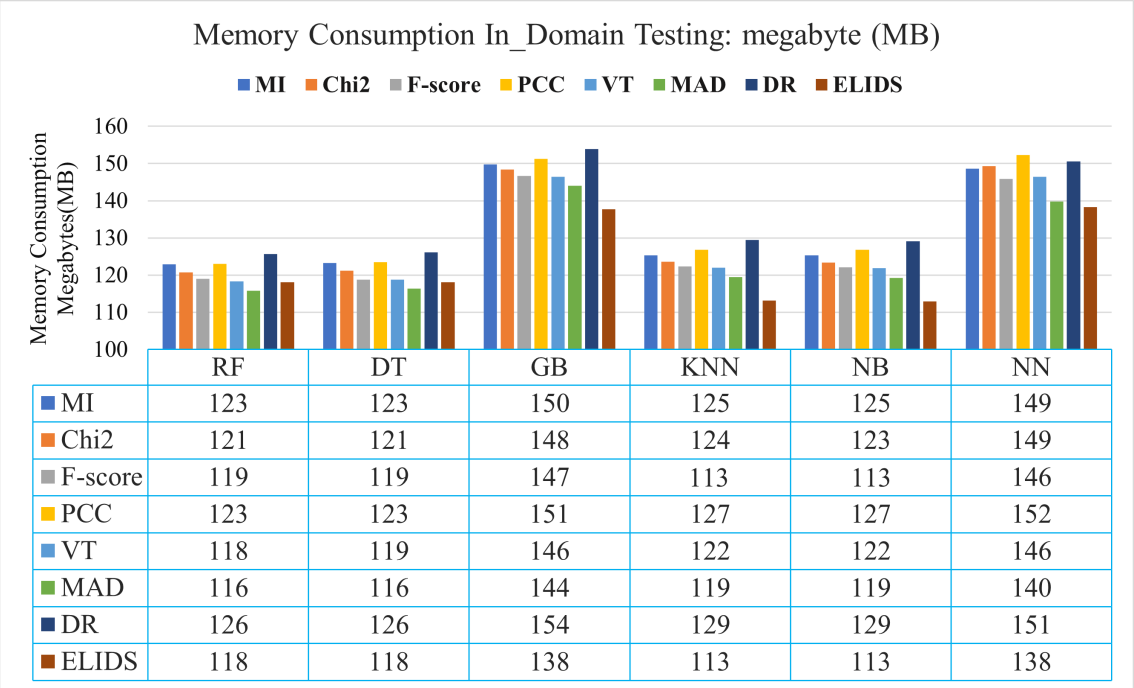


	RF	DT	GB	KNN	NB	NN
MI	1049	1050	1064	1121	918	1208
Chi2	928	928	942	999	797	1086
F-score	833	833	847	904	702	991
R	846	847	861	918	715	1005
VT	846	846	860	917	715	1004
MAD	862	863	877	934	731	1021
DR	1282	1282	1296	1353	1150	1440
ELIDS	1430	1430	1442	1479	1440	1452

Figure 4.6: Memory required in training ML classifiers

Figure 4.7 illustrates memory consumption during the testing phase, where results show that for all the six trained ML models, DR FS technique exhibits the highest memory consumption. Results show that ELIDS based classifiers exhibit the lowest (i.e. for case of GB, KNN, NB and NN) or the second lowest (i.e. for

case of RF and DT) memory consumption requirements as compared to competitor classifiers build using the individual FS techniques. Even when ELIDS RF and DT based classifiers fall next to MAD based RF and DT classifiers, a marginal difference is observed in memory consumption. ELIDS classifiers consume 118 MB each for RF and DT, while MAD classifiers takes 116 MB each for RF and DT. Results exhibited by ELIDS based classifiers show promising performance as memory consumption. The classification models are below 140 MB which can be deployed over many resource-constrained IoT devices, such as IoT Gateways and IoT End Devices.

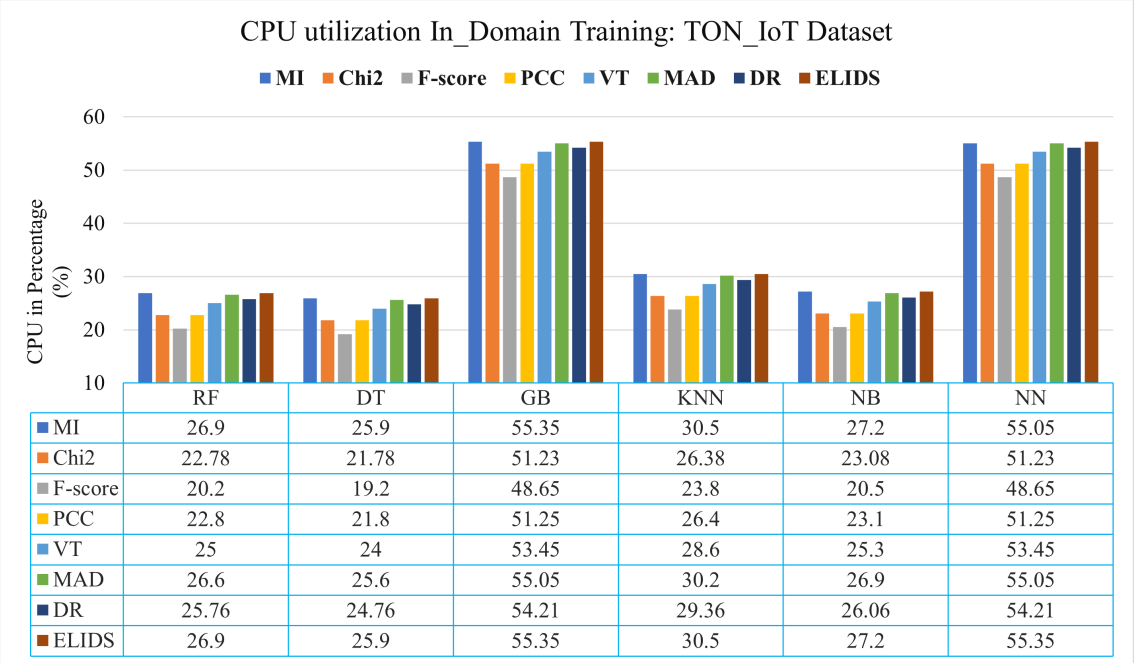


	RF	DT	GB	KNN	NB	NN
MI	123	123	150	125	125	149
Chi2	121	121	148	124	123	149
F-score	119	119	147	113	113	146
PCC	123	123	151	127	127	152
VT	118	119	146	122	122	146
MAD	116	116	144	119	119	140
DR	126	126	154	129	129	151
ELIDS	118	118	138	113	113	138

Figure 4.7: Memory required in testing ML classifiers

The CPU utilization at the time of the models training phase is presented by Figure 4.8, It shows that MI and ELIDS based classifiers consume the most CPU resources during the training phase. MI's higher CPU consumption can be

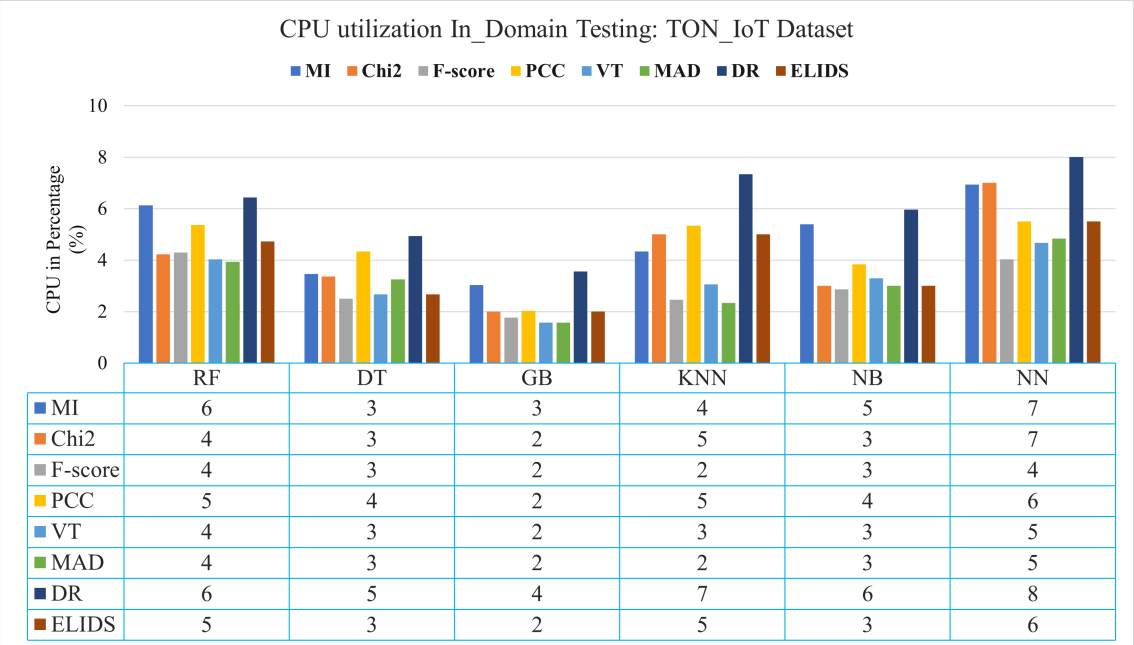
attributed to its information-theoretic calculations and pairwise feature-to-target variable comparisons, especially in high-dimensional datasets. On the other hand, the proposed ELIDS is an ensemble method that utilizes CPU resources during the execution of various FS techniques that act as building blocks for this approach. In ELIDS, the FS techniques are run sequentially, so the CPU utilization in the proposed method is determined by the maximum CPU used by any of the techniques. It is also worth highlighting that F-score based classifiers result in the minimum CPU utilization across the entire spectrum of the six used ML algorithms. Having said that, the difference in CPU resource utilization between ELIDS based classifiers and F-score based classifiers is small. As the training process takes place over resourceful devices, hence the issue of slightly higher CPU utilization exhibited by ELIDS based classifiers can be viewed as trivial.



	RF	DT	GB	KNN	NB	NN
MI	26.9	25.9	55.35	30.5	27.2	55.05
Chi2	22.78	21.78	51.23	26.38	23.08	51.23
F-score	20.2	19.2	48.65	23.8	20.5	48.65
PCC	22.8	21.8	51.25	26.4	23.1	51.25
VT	25	24	53.45	28.6	25.3	53.45
MAD	26.6	25.6	55.05	30.2	26.9	55.05
DR	25.76	24.76	54.21	29.36	26.06	54.21
ELIDS	26.9	25.9	55.35	30.5	27.2	55.35

Figure 4.8: CPU utilization in training ML classifiers

CPU utilization at the time of models testing is presented by Figure 4.10, which reveals that DR based classification models exhibit the highest CPU demand across all six ML algorithms. In contrast, ELIDS based classifiers do not exhibit the least amount of CPU utilization but show small CPU utilization as low as 2.0% with GB based classifier, and as high as 6% with NN based classifier. This shows the suitability of running ELIDS based classifiers for IoT resource constraint devices.



	RF	DT	GB	KNN	NB	NN
MI	6	3	3	4	5	7
Chi2	4	3	2	5	3	7
F-score	4	3	2	2	3	4
PCC	5	4	2	5	4	6
VT	4	3	2	3	3	5
MAD	4	3	2	2	3	5
DR	6	5	4	7	6	8
ELIDS	5	3	2	5	3	6

Figure 4.9: CPU utilization in testing ML classifiers

The results presented in Figures 4.2 to 4.9 for in-domain evaluations reveal that resource consumptions of ELIDS-based classifiers are efficient in both phases of training and testing. ELIDS exhibiting a relatively higher resource requirement at the training phase is acceptable in our scenario as the training is an offline process that generally takes place over resourceful devices, and the same does not affect resources required at the time of testing/deployment. Moreover, the proposed ELIDS model is lightweight and does not impose significant memory and CPU overhead

when it comes to its execution at the testing phase. ELIDS takes This characteristic makes ELIDS-based classification models a suitable approach, particularly for resource-constrained IoT devices. The results indicate that the ELIDS-based classifiers are highly effective in detecting DoS/DDoS attacks, achieving a maximum accuracy of 99.8%. The ELIDS system utilizes 0.000625 ms to test a single sample, 0.15 MB of memory, and 0.0006% CPU with the RF classifier.

4.3.4 Phase III: Cross-domain Evaluations for Testing

This section provides another but more rigorous set of evaluations that depicts cross-domain testing for the classification models built using the proposed ELIDS approach and the conventional individual FS techniques. In these evaluations, a different set of testing samples is taken from BoT-IoT dataset, as is a dataset originating from a different domain, as compared to the dataset through which the classification models were originally trained. The trained classification models integrated in an IDS and deployed in real-world scenarios have a strong potential to generate false alarms, especially when exposed to unseen network traffic, generally caused due to model overfitting. To largely mimic a scenario of having real-world and unseen network traffic applied over the classification models under examination, the cross-domain evaluations approach helps in validating such a scenario. In this section, only evaluations in the testing phase of the classification models are presented as evaluations of the training phase are presented and discussed previously in Section 4.3.3

Since another publicly available dataset known as BoT-IoT is being employed

for executing the testing phase, data preprocessing is executed over the underlying dataset which contains the same process as described previously in the proposed methodology. Additionally, for cross-domain testing, the same testing sample size is opted for, i.e. 8,000 samples, as was the case for in-domain evaluations. While doing so, The process also includes feature alignment, a critical step to ensure consistency in the datasets. In this step, seven features are carefully selected from the BoT-IoT dataset that correspond to the seven features used to train the classification models. This alignment is essential because discrepancies in feature sets between datasets can lead to misleading performance evaluations. ELIDS and individual FS-based classifiers are evaluated by measuring various performance factors, including detection accuracy, performance metrics, testing time, memory consumption, and CPU utilization.

Figure 4.10 presents the accuracy performance of the ML Classifiers for ELIDS and individual FS-based methods. The evaluation results show a significant decrease in the performance of FS-based classifiers when tested with a cross-domain dataset compared to their performance in in-domain evaluations. However, the graph also depicts that ML classification models trained using the proposed ELIDS approach are more robust towards a change in data patterns and less prone to overfitting, hence largely surpassing in accuracy performance when compared with the competitors. The ELIDS based classifier trained with the RF algorithm represents the highest accuracy of 69.8%, which exceeds by at least 23.8% the individual FS-based classifiers prepared using the same RF algorithm.

Among all ELIDS based classifiers, the NB classification model produces the

lowest accuracy. This is observed due to the classifier's reliance on probabilistic calculations and hence being susceptible to outliers, where outliers in the unseen data can impact the model's ability to make accurate predictions. Moreover, the NB classifier makes strong independence assumptions between features, which may not be held in all datasets. Hence, if the independence assumption is violated, the model's accuracy can be negatively affected as well. Having said that, ELIDS based naïve classifier still outperforms other individual FS base naïve classifiers.

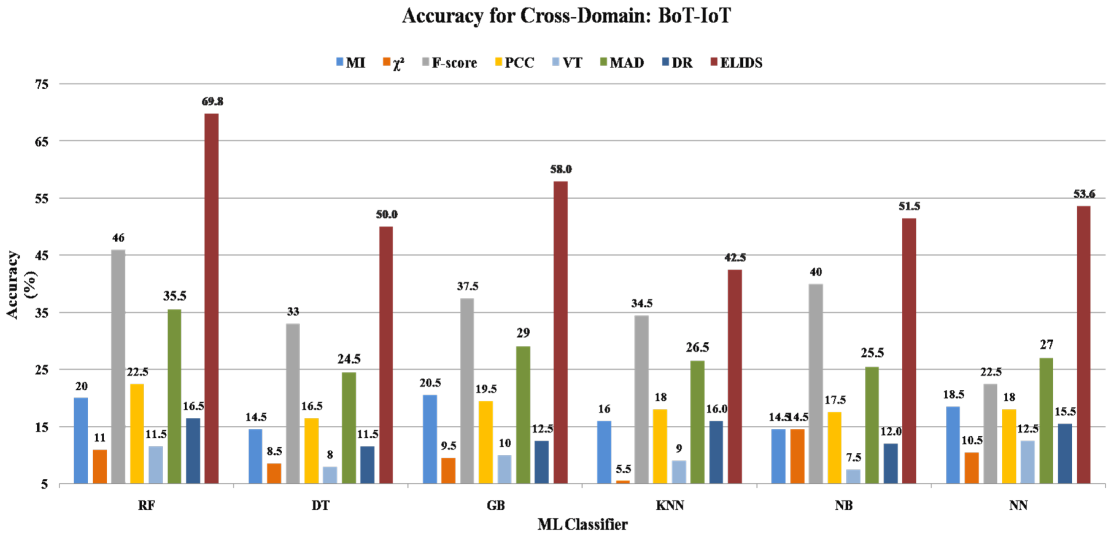


Figure 4.10: Accuracy graph of trained ML models tested on BoT-IoT dataset

Figure 4.11 (a, b, c, and d) presents the performance metrics for TPR, TNR, FNR, and FPR, respectively. In terms of successfully identifying DDoS data as an attack, the TPR of the individual FS approaches is exceptionally low as shown in Figure 4.11(a). However, the proposed ELIDS based classifiers using RF and NN algorithms achieve a peak TPR of 0.6 and 0.54, respectively. On the other hand, values obtained for FNR are depicted by Figure 4.11(c), which are in complement to those obtained in TPR. The skyrocketing FNR values for all individual FS-based

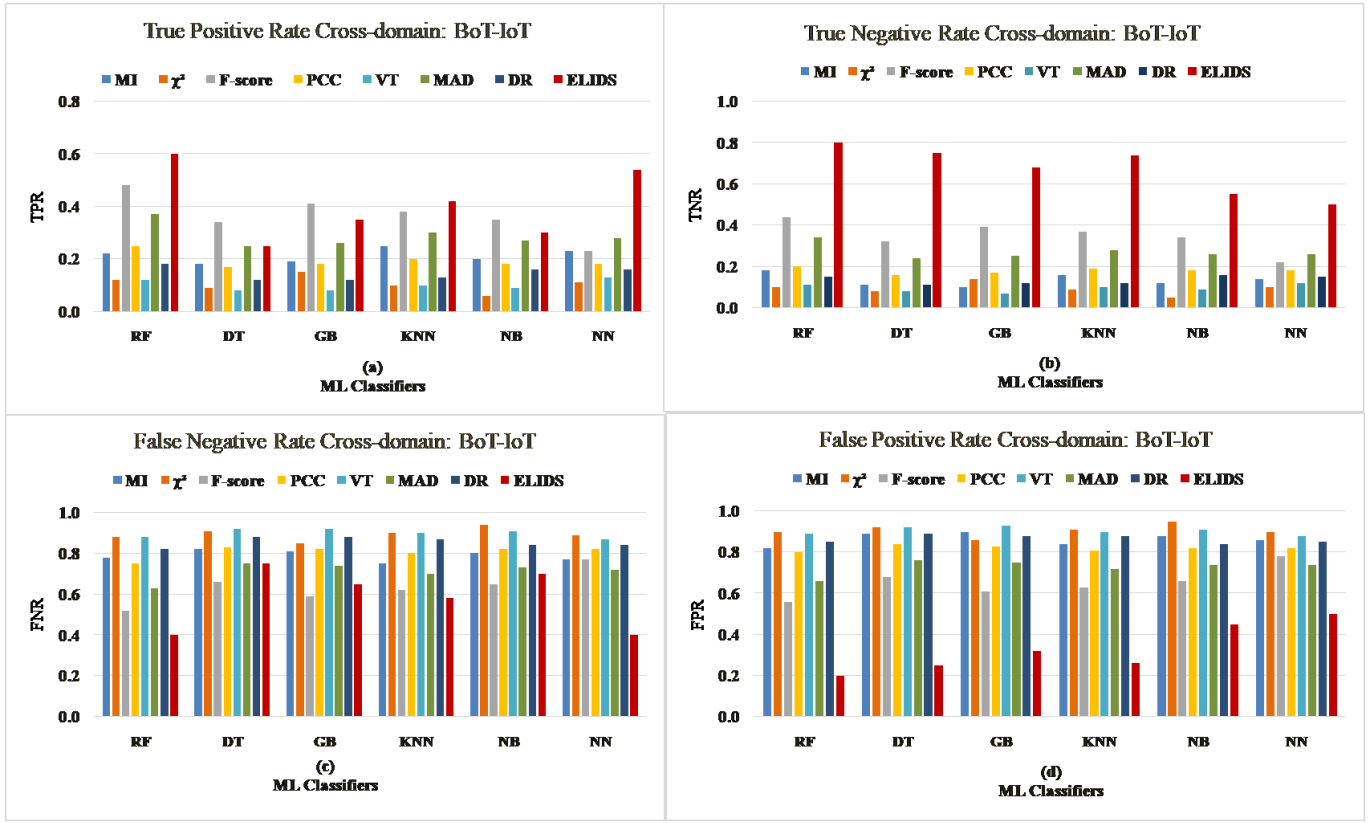


Figure 4.11: Performance metrics (a) TPR (b) TNR (c) FNR and (d) FPR

classifiers indicate the failure of these classifiers by identifying DDoS attack traffic as normal traffic. However, the proposed ELIDS based classifiers exhibit similar failure behavior except for the case of RF and NN-based classifiers as they exhibit misdetection rates of 0.4. Results in Figure 4.11 (a and c) indicate how ELIDS based classifiers with RF and NN as the learning algorithms are robust to changes occurring in the testing data patterns, which is not the case for all other tested classifiers.

ELIDS based classifier obtains a TNR value of 0.8 for RF, while also achieves reasonable TNR values for DT, GB, KNN, NB, and NN, as all depicted by Figure 4.11(b). In contrast, classifiers based on the conventional FS methods fall in performance by obtaining relatively low TNR values as compared to those obtained by ELIDS based classifiers. On the other hand, classifiers based on ELIDS and the individual FS techniques show FPR in complement to what is obtained in TNR. As shown in Figure 4.11(d), the RF-based ELIDS classifier achieves an FPR value of 0.2. The results indicate that ELIDS-based classifiers can correctly identify normal traffic while generating fewer false alarms. Therefore, ELIDS-based classifiers outperform other individual FS-based classifiers in detecting normal network traffic.

In the context of cross-domain testing, evaluation of testing time and resources usage has been primarily focused on the ELIDS-based models. This focus is justified because ELIDS-based classifiers consistently demonstrate substantial accuracy and robust confusion matrix performance across all evaluated ML models. Although individual FS-based classifiers are trained on the same seven features, their resource usage is expected to be nearly identical, and their lower predictive performance

makes a separate detailed resource analysis less informative. Therefore, the analysis emphasizes ELIDS, which represents the most effective and efficient approach among the evaluated models.

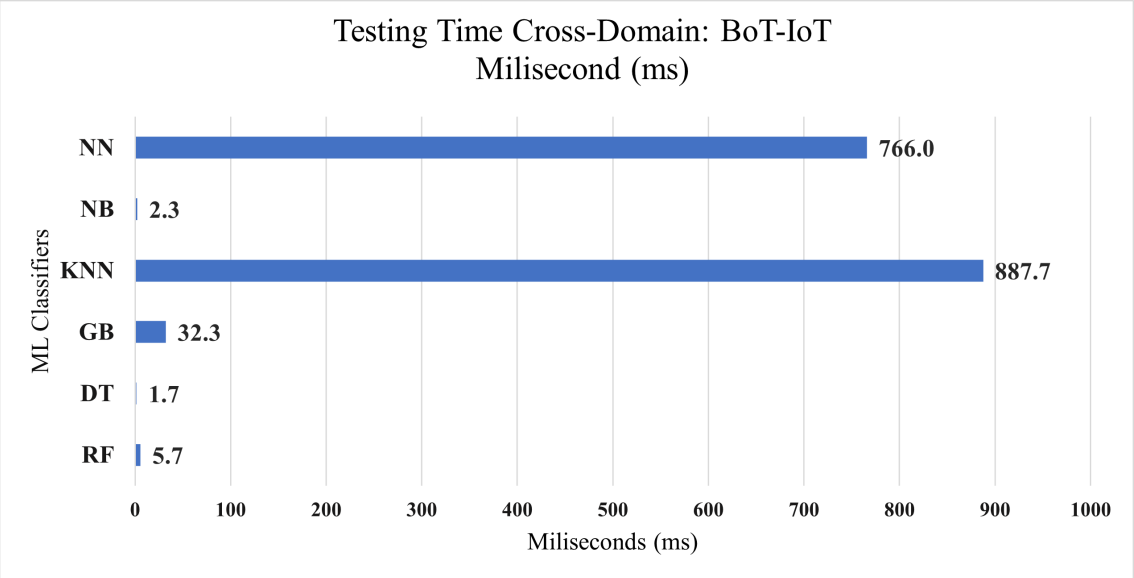


Figure 4.12: Testing time required by ML classifiers using ELIDS

Figure 4.12 presents the testing time required to make predictions on 8,000 BoT-IoT data samples while considering the ELIDS based classification models. In the context of testing time, both NN and KNN-based classifiers require significantly more time for testing the considered samples, which is a result of their computationally intensive nature. However, the ELIDS based classifiers perform best with RF, DT, GB, and NB classifiers with a relatively low testing time of 5.7, 1.7, 32.3, and 2.3 msec, respectively.

Moreover, Figure 4.13 depicts both the memory and CPU consumption by the ELIDS based classification models. In terms of memory consumption, ELIDS based classifiers show relevantly low memory requirements while considering RF, DT, GB,

and NB as the learning algorithms. In contrast, high memory consumption is exhibited when employing KNN and NN for building classification models. In terms of CPU usage, ELIDS based classifiers consume relatively low CPU resources (i.e. as low as 2%) when employing RF, DT, KNN, and NB as the learning algorithms. Conversely, higher CPU resources of 7% and 6% are observed by ELIDS based classifiers when employing GB and NN as the learning algorithms, respectively.

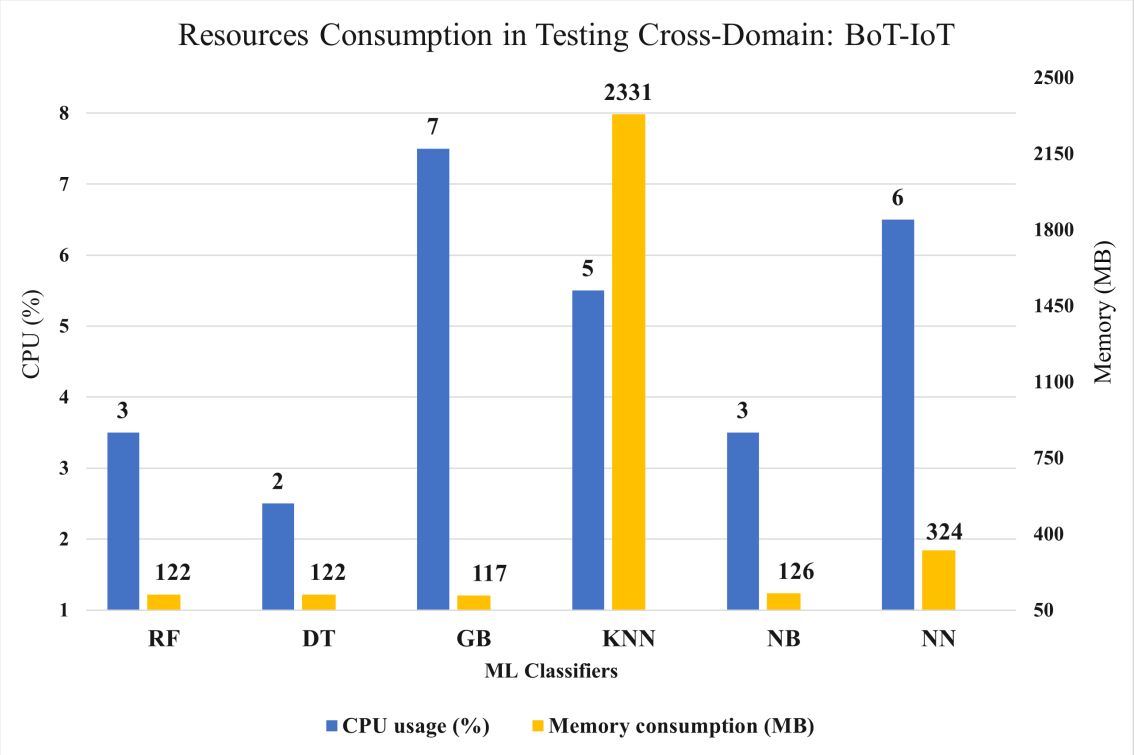


Figure 4.13: CPU and memory utilization in testing ML classifiers using ELIDS

The results presented in Figures 4.10 to 4.13 for cross-domain evaluations reveal that several ELIDS-based classifiers are efficient in terms of training time, memory consumption, and CPU usage. As a result, such classifiers can be considered lightweight and suitable for deployment over resource-constrained IoT devices. The results also show that the ELIDS-based classifiers outperform the existing individual

FS-based classifiers by large margins in terms of accuracy and confusion matrix parameters. In particular, the ELIDS-based classifier utilizing the RF algorithm demonstrates the most promising cumulative performance, achieving an accuracy of 69.8% with TPR and FNR values of 0.57 and 0.43, respectively. Moreover, RF requires only 0.15 MB of memory, 0.00037% CPU, and 0.0007 ms to test a single sample, making it the most suitable combination of the proposed ELIDS approach with an ML algorithm.

4.4 Summary

In this chapter, the performance and resource efficiency of the proposed ELIDS method using six different Machine Learning algorithms are analyzed and contrasted with seven different conventional FS techniques based on filter-based classifiers. In the in-domain analysis, it has been observed that ELIDS-based as well as conventional FS-based classifiers are highly efficient in identifying DoS/DDoS attacks with a maximum accuracy of 99.8% for various classifiers. In contrast to in-domain analysis, the cross-domain analysis proves the dropping efficiency of conventional FS-based classifiers in terms of accuracy and detection rates with a considerable drop in performance factors in terms of accurately identifying various attack patterns in ELIDS-based classifiers. Moreover, it has been observed in this analysis that ELIDS-based classifiers have a substantial edge in performance and efficiency measures in comparison to conventional FS-based classifiers.

In both evaluation tasks, the classifiers based on ELIDS consumed lower memory and CPU cycles during the testing process. Among these classifiers, the one

based on ELIDS with the RF algorithm shows the best performance and cost-effectiveness. In this respect, this classifier shows the highest accuracy of 69.8%, outperforming all other traditional FS-based classifiers by at least 23.8%.

The analysis of evaluation outcomes clearly signifies that ELIDS performs better in comparison with the latest approaches mentioned in Table 2.6. Based on resource usage, in comparison with [126], [112], and [123], in testing 7,000 samples, our proposed system consumes only 118 MB of memory and 5% CPU usage, whereas other authors have mentioned usage of CPU in GHz and consumption of memory in GB. For cross-domain analysis, comparison is not feasible since these authors have not analyzed their proposed models in a cross-domain setting. These features are strongly recommended by a single technique and are neither redundant nor consistently recommended by other techniques. As simulated experiments demonstrate, deletions of such features do not affect the performance. It enhances the generalizability of the features and prevents overfitting. It clearly illustrates the effectiveness of the ensemble technique in making a good balance between feature importance and feature reduction, selecting only features that are consistently important to all techniques. It clearly demonstrates that classifiers based on EL IDS perform better than classifiers based only on FS techniques. Their performance drops when tested with new samples. Keeping in view these aspects, the following chapter proposes a self-healing technique to increase the adaptability and robustness of IDS in dealing with new samples.

CHAPTER 5

Proposed SHIoT Mechanism for ML-Enabled IDS

5.1 Introduction

The current chapter outlines the process used to integrate self-healing features into a machine learning-enabled intrusion detection system (IDS) in an Internet of Things (IoT) setting. The design incorporated in the newly suggested Self-Healing Internet of Things (SHIoT) system follows a multi-stage approach consisting of a set of key steps that focus on enhancing its ability to automatically identify and react to emerging DoS/DDoS threats. These key steps include monitoring, packet capturing, clustering, and re-training the machine learning-based IDS model.

In addition, the chapter explains the meaning of the threshold calculation that has a critical part in distinguishing between normal and abnormal resource

consumption in an attacked IoT device and a normally operating IoT device. Upon exceeding the resource threshold value, the device sends a danger signal that triggers a self-healing procedure that ends with the application of a new IDS model. The SHIoT technique was designed to improve the resistance of IoT systems and ensure that the IDS can heal itself when attacked.

5.2 Self-healing in IoT Systems

"Self-healing" refers to the ability of a system to automatically detect and correct itself in the event of an interruption or failure, thereby enabling it to operate continuously [166]. Like the process of healing in biology after injury, whereby the skin regenerate [167], in computing, "self-healing" refers to the ability of a system to automatically detect, diagnose, and correct an anomaly or an interruption to ensure that it continues to run in an optimal manner [166, 168]. According to [169], self-healing systems play an important role in encouraging resiliency in complex systems that may require intensive manual intervention in a distributed network like that of IoT, or even in huge data centers.

Among these emerging research areas, it can be observed that self-healing in IoT-based networks has gained widespread interest in recent years due to rising cyber threats. In this environment, self-healing attack detection, encompassing intrusion detection systems (IDS), signifies the ability of an IDS to self-detect and recuperate from malicious attacks or irregular patterns [170]. As opposed to normal IDS mechanisms, which only notify users of malicious traffic, a self-healing IDS has the adaptability and capability to modify and recuperate affected functionality in

response to rising cyber threats [170]. Self-healing in this context is highly necessary in a network like IoT, which requires instantaneous and self-reliant mechanisms [170].

5.3 Proposed Self-Healing Mechanism for IoT

As shown in Figure 5.1, the proposed SHIoT method includes four different steps, namely, Step I – Health Monitoring, Step II – Packet Collection, Step III – Retraining, and Step IV – Evaluation. For Step I, health monitoring related to the utilization of IoT devices is performed, along with the management of danger signal production. For Step II, danger signals, as well as network packet reception, are addressed. For Step III, clustering of network packets, along with the process of model retraining, takes place.

Finally, the Step IV mainly targets the assessment of the proposed SHIoT mechanism effectiveness. The assessment herein involves KPM parameters like accuracy, CPU usage, and memory usage, as well as the confusion matrix, which plays a crucial role in system efficacy evaluation. Notably, the IoT devices involved here remain resource-limited; hence, they lack adequate computing resources like memory and CPU power [52]. These resources can barely meet tasks like data processing and the dissemination of sensed data among targeted devices [171]. In normal usage, the resource usage among devices like those of the IoT category remains low or moderately high; however, during a DoS/DDoS attack, the attacker targets these very same computing resources through flooding of the device with a high number of malicious requests. Based on the rising number of requests, the CPU and memory

usage remain abnormally high as a consequence of responding; this eventually culminates in 'resource exhaustion' through which the device becomes non-functional or non-accessible.

The monitoring of resource usage can analyze anomalies and be used as an early warning system in an attack [172]. Considering that resource-constrained devices exist in the IoT ecosystem, Figure 5.1 starts with two main components at Step I. Both components exist inside the IoT device and are at the perception layer. The first component consists of two hardware components (CPU and memory) and a network component (packet rate). As this work is focused on detecting network-layer DoS/DDoS attacks, packet rate becomes an essential component. This is because it analyzes the data inputs and outputs from/to the IoT device. This allows the tracking of health and the detection of an attack on the system. Additionally, the second system, name "health monitoring" system, runs in all the design's IoT nodes. The system undertakes the constant assessment of the use of all resources by monitoring CPU usage, memory usage, and packet rate. The assessment is a comparison of all resource usage against set threshold levels. If resource usage exceeds these levels, it sends an alert to the health monitor, called a "danger signal." This signal serves as an initial sign of an attack on the system, aimed at sensitizing the system to initiate its healing process. The whole healing process, as described in the figure 5.1, denoted by figures 1 to 6, implies the whole healing process's detailed description in the subsequent sections.

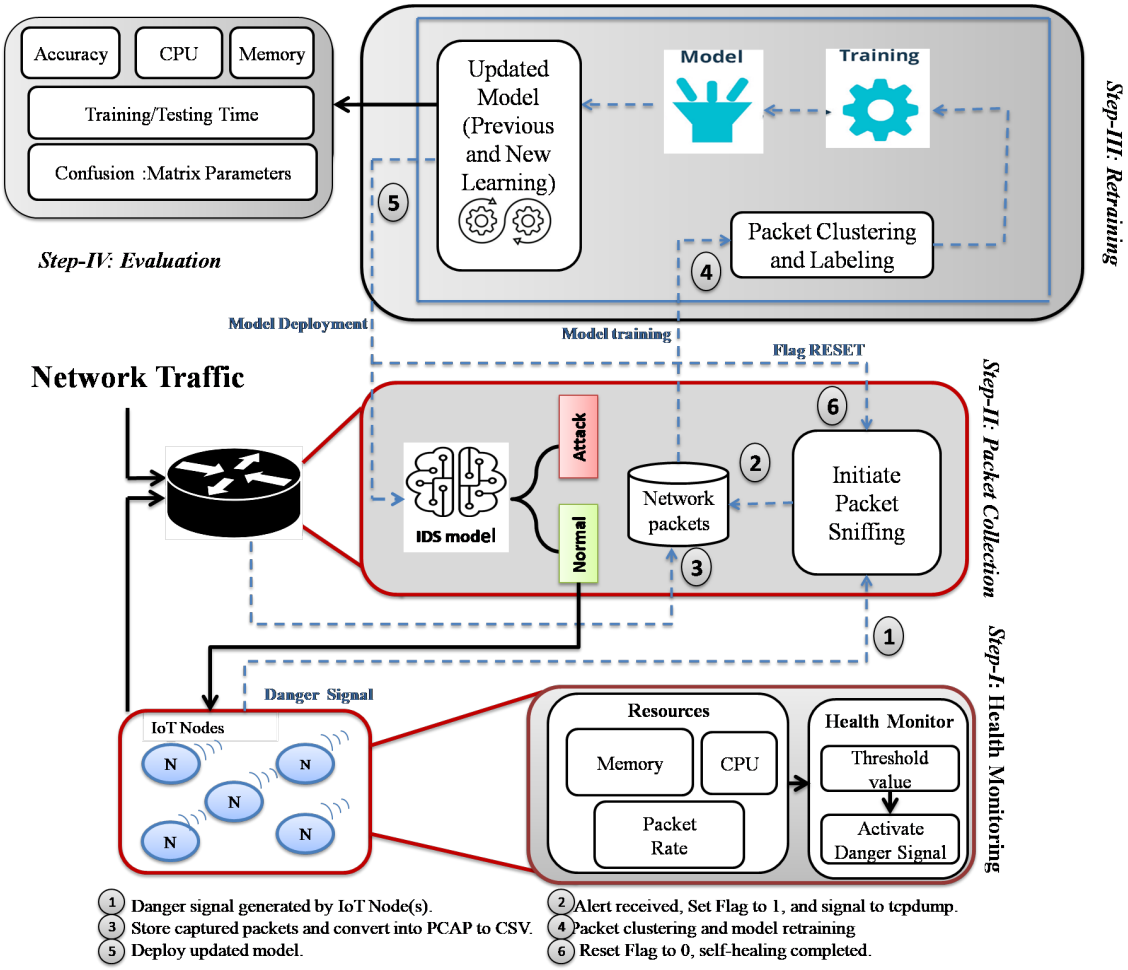


Figure 5.1: Proposed self-healing mechanism for IDS

5.3.1 Calculation of Threshold Value

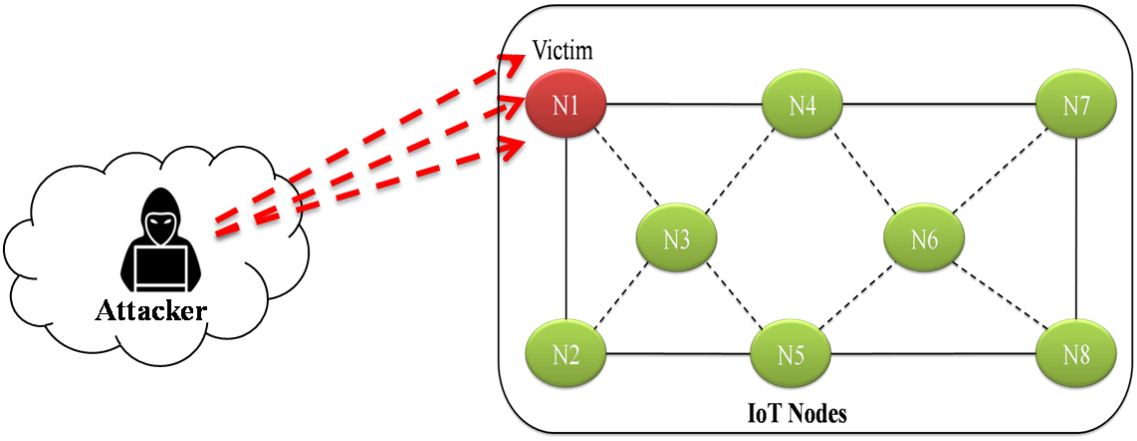
These threshold values are calculated using a large number of experiments that examine the activity of an Internet of Things (IoT) device under both standard and attack situations. In this laboratory setup, Virtual Machines (VMs) are used to simulate IoT devices with limited processing capabilities, mimicking a realistic scenario in a physical environment, listed in detail in Table 5.1. For each VM, one CPU core is allocated, with a 512 MB memory capacity and a reduced Raspbian operating system. For wireless communication between devices, this laboratory setup adopts

a Wi-Fi 802.11ah network, specifically designed with low power consumption and a longer communication range, making it more suitable and appropriate for a smart environment (Lee et al., 2021). In addition, each virtual IoT device has a python script running in the background with a smart temperature sensor functionality that pings temperature values every second from one IoT device to a gateway. In this standard scenario, this virtual IoT device consumed only 4-5 percent of CPU usage and 194-199 MB of system RAM in processing four incoming and four outgoing packets per second (pps).

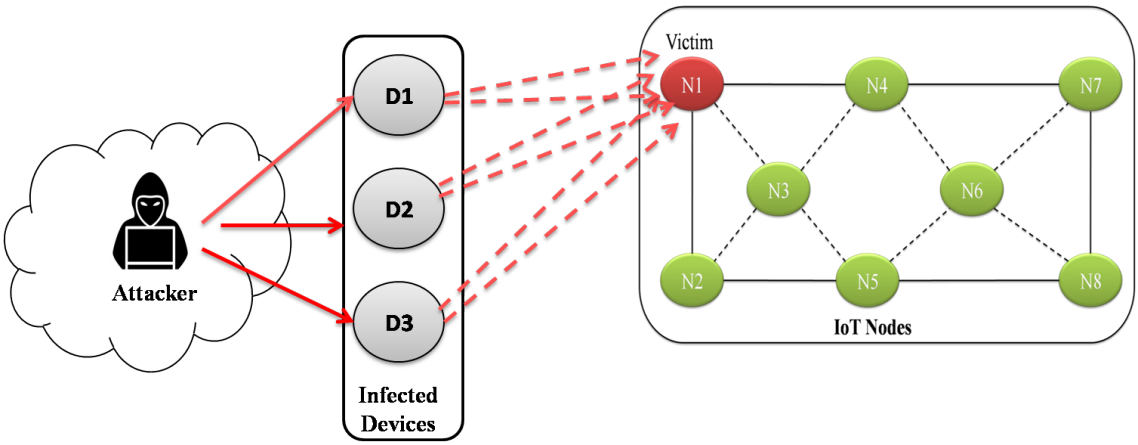
Table 5.1: Configuration details of the smart temperature sensor device

Resource	**Configuration**
Device Name	Smart Temperature
Memory	512 MB
Processor	Single-core, ARM architecture, 1.2 GHz
Operating System	Raspbian Lite
Wireless Connectivity	802.11 ah

The topology used for launching DoS/DDoS attack against the IoT device is shown in Figure 5.2(a) for DoS attack and Figure 5.2(b) for DDoS attack. For the DDoS attack, shown in Figure 5.2(b), it involves a C&C server controlling a set of compromised machines designed to flood the victim system with network traffic in order to make it unavailable to its desirable users. For DoS/DDoS attack engagements, the Kali Linux framework is used alongside tools like HPING3 [173], Metasploit [174], and LOIC [175], for sending ICMP, UDP, and TCP traffic.



(a) DoS attack



(b) DDoS attack

Figure 5.2: Attack topologies

After the commencement of a DoS/DDoS attack, there are observable effects on the IoT device and its resources. The effects are discussed in relation to the operational capacity of the device in this topic as it pertains to increased resource usage and system failures.

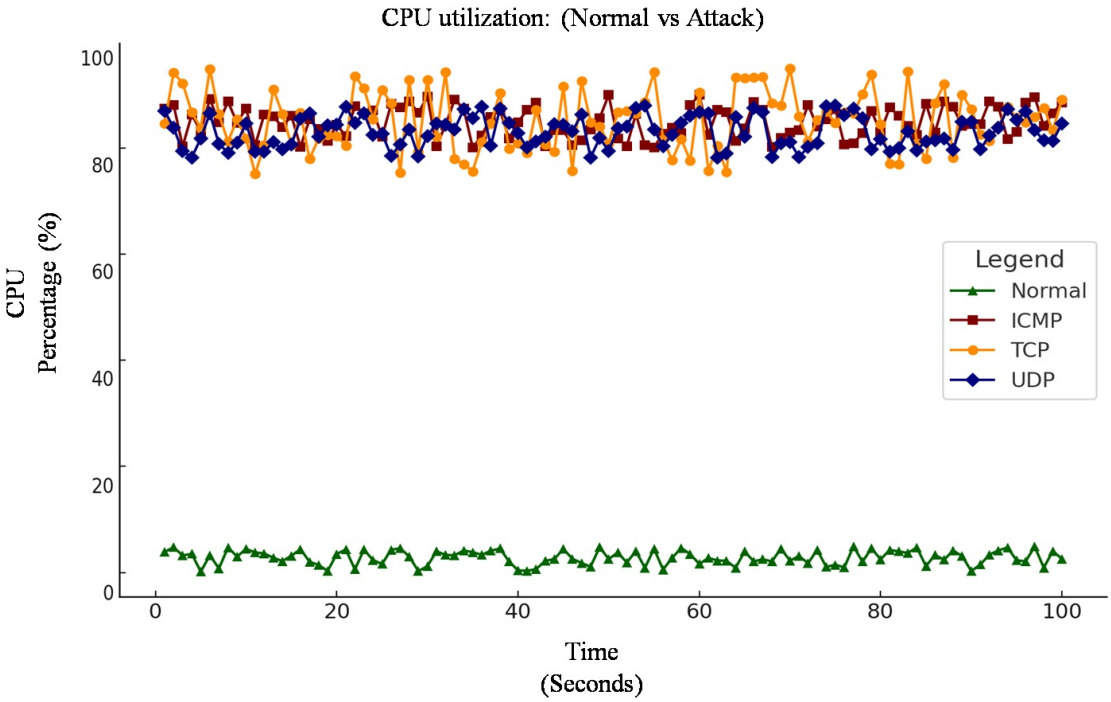


Figure 5.3: CPU utilization during normal operation and DoS/DDoS attack

Figure 5.3 illustrates the CPU utilization of an IoT node when it works in a normal manner and when it encounters DoS/DDoS attacks. In the normal situation, the CPU utilization of the IoT device is relatively low and falls between 1.7% and 4.4%. These values indicate normal usage of resources for performing usual functions such as gathering data, processing the data, and communicating with neighboring devices and a gateway. These activities require less computational power and lead to the shown low CPU usage, which typically represents normal IoT devices when there are no attacks.

By contrast, under attack traffic, CPU usage in the compromised node significantly increases and stays between 79% and 89%. In the case of TCP and UDP flood attacks, this is because the compromised node receives an overwhelming num-

ber of inappropriate requests, such as synchronization, handshaking, or establishing connections. In ICMP flood attacks, this is because the node receives a tremendous number of echo requests, or pings. Such increased traffic means that more time is needed for packet validation and connection establishment, leading to a tremendous jump in CPU usage. Further, this increased CPU usage significantly burdens the processing mechanisms in the node, leading to underperformance.

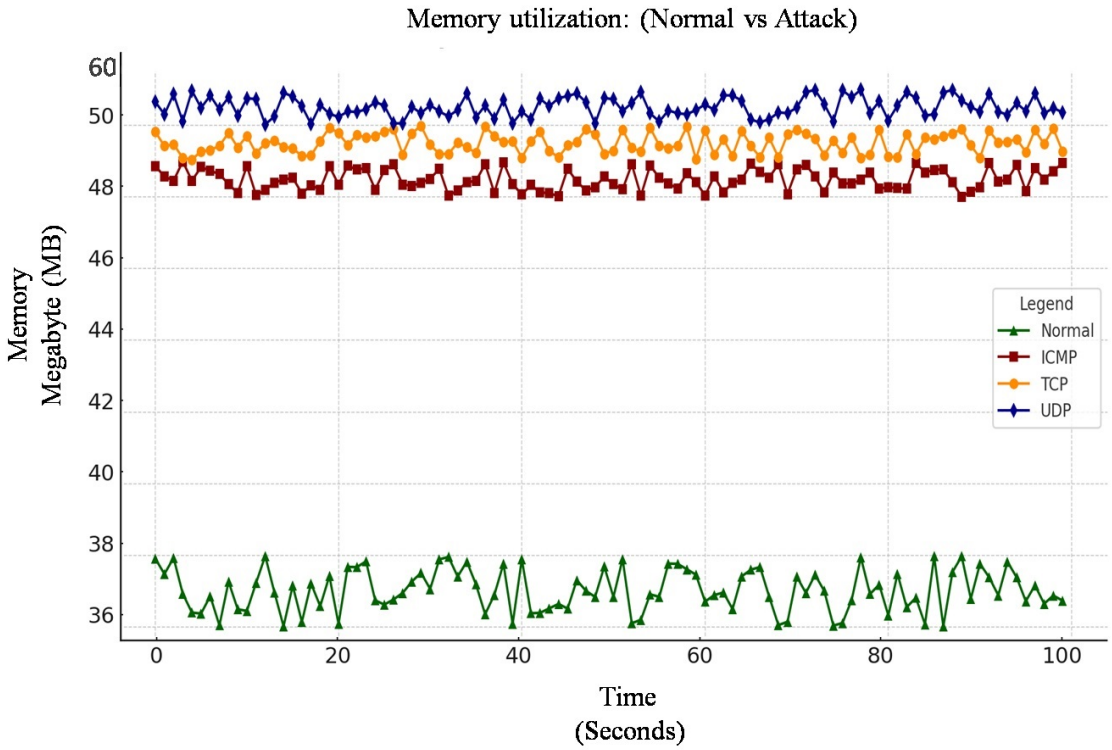


Figure 5.4: Memory utilization during normal operation and DoS/DDoS attack

From the figure above (Figure 5.4), the memory use of an Internet of Things (IoT) node when running in the normal state and under the three flood attacks (ICMP, TCP, and UDP) is shown. The graph illustrates the variability in memory use between the two states. The baseline memory usage lies in a stable state with values fluctuating from 36 MB to 37.9 MB, indicating the baseline memory require-

ments of the IoT device to perform the normal tasks. As the memory is subjected to an attack, the memory usage rises with increased allocations to deal with the packets.

For instance, memory usage during the TCP attack attains a peak of 50.4 MB. Additionally, during the ICMP and UDP attacks, memory usage attains a peak of 49.7 MB and 60 MB, respectively. In addition, as shown in Figure 5.4 below, memory usage during the UDP attack outpaces that of the TCP attack by 9.6 MB and outpaces the ICMP attack by 10.3 MB. The high memory usage during the UDP attack can be related to the fact that the UDP attack is a connectionless type of attack, whereby the attacker sends a constant flow of packets without waiting for acknowledgments. As a consequence, this forces the victim node to handle a large number of packets arriving from the attacker, thus resulting in a high memory requirement.

The graph in Figure 5.6 above illustrates the number of packets the IoT node receives during the normal and attack scenarios. During the normal situation, the IoT node receives four packets. This indicates that the IoT node is processing the requests optimally without overburdening the resources. Contrary to the normal situation, the IoT node adopts a drastically different approach while handling the increased load during the attack. For instance, during the TCP attack, the number of packets increases to 20,159-23,760 pps.

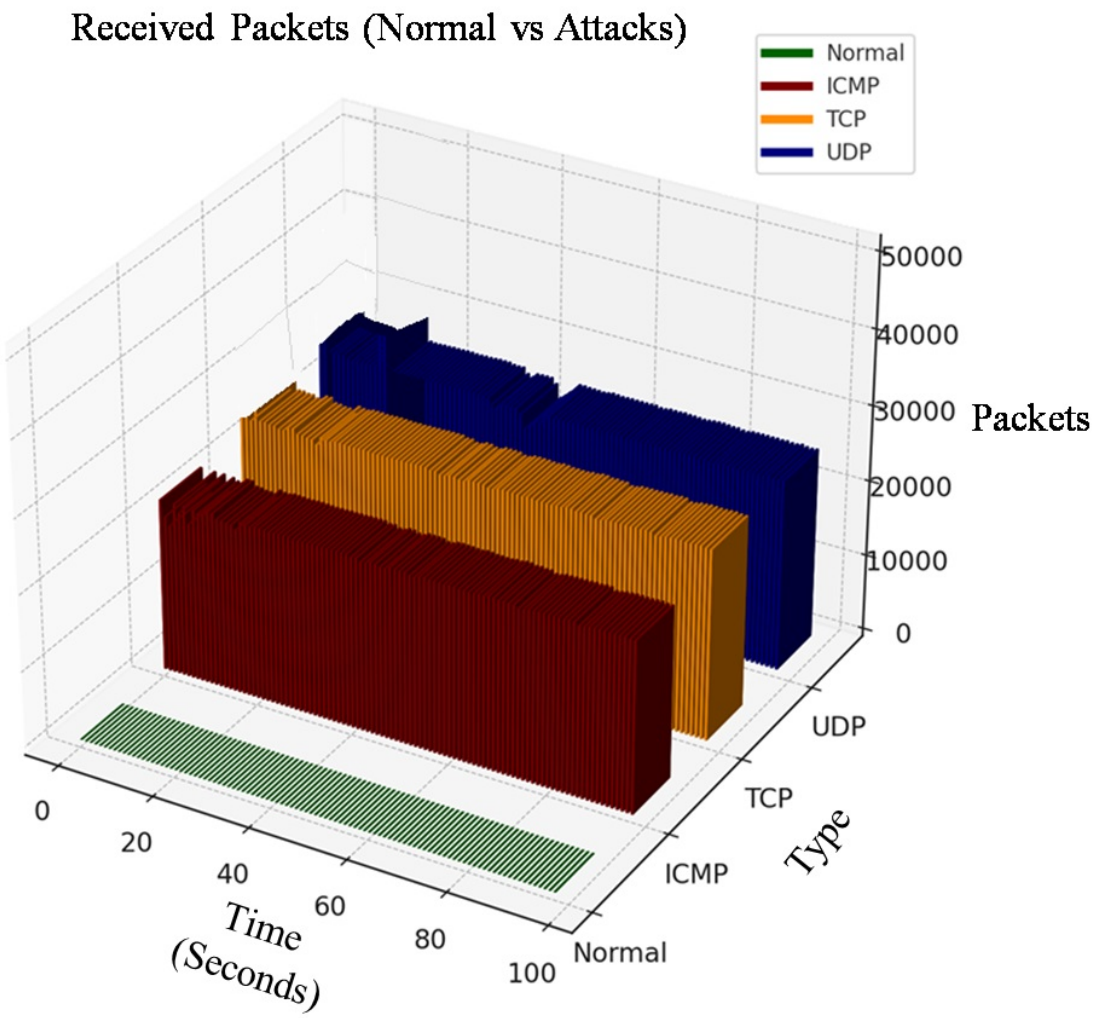


Figure 5.5: Packets received during normal and DoS/DDoS attack

This drastic change in the number of incoming packets forces the system to deal with a significantly higher number of packets than what was intended, putting a lot of pressure on the system resources available at the node. In a UDP attack, the packet flow rate varies between 20,731 and 25,000 pps, whereas during the ICMP attack, it varies between 19,681 and 22,910 pps. The higher packet flow rate during the attack period confirms the allocation of a large number of system resources at the attacking node to deal with the large number of incoming packets during the attack. As a consequence of such increased resource utilization related to CPU usage,

memory storage, and bandwidth usage at the node during the attack situations, the system operation at the node gets hampered, which may further result in the failure of the system at the node.

Based on the details provided in Figures 5.3 to 5.5, showing the resource usage of the IoT node encompassing CPU, memory, and packet rate, as well as the resource usage patterns of the IoT node under normal conditions as well as DoS/DDoS attack conditions, the value of the threshold for each resource type is calculated by adding up the highest resource usage under the normal condition and the lowest resource usage under DoS/DDoS attack. By calculating the midpoint of the two extreme resource usage values, the danger signal threshold is systematically set up with the aim of triggering danger signal alerts when the resource usage exceeded the set threshold value [176]. Furthermore, the calculated midpoint represents the boundary point of the danger zone with the least risk of misclassifying the attack scenario within the context of attack/normal conditions sufficiently apart in terms of resource usage patterns [177]. It is reliable, computationally efficient, and provides precise danger signal thresholds without exerting extra resource demands on the IoT node. Further, the algorithm is simple and can be easily implemented. Being an algorithm that does not call for complex processes in data analysis or machine learning, it is most ideal and useful in applications that call for instant decision-making. In a scenario such as an IoT network, where computation is limited, this algorithm provides a most optimal trade-off between performance and resource optimization. Therefore, this algorithm provides a most reliable threat detection system that is free from impacting performance.

CPU: The threshold value for the CPU is determined by averaging the maximum (Max) CPU usage observed during normal operational conditions and the minimum (Min) CPU usage recorded during an attack, shown in equation 5.1. This method establishes a critical threshold for distinguishing between standard and potentially malicious activity. The threshold is calculated using the following Equation (5.1):

$$T_{\text{CPU}} = \frac{\max(\text{CPU}_{\text{Normal}}) + \min(\text{CPU}_{\text{Attack}})}{2} \tag{5.1}$$

The CPU threshold is calculated by averaging the maximum CPU utilization observed during normal operation (4.4%) and the minimum CPU utilization recorded during an attack (79%), given in Figure 5.3, resulting in a threshold of 41.7%. This threshold is a decision boundary for distinguishing between normal and attack scenarios. It is noteworthy that the calculated CPU usage threshold T_{CPU} of (41.7%) is adaptive only at the system setup time. Although the nature of the IoT network is heterogeneous, the threshold values are determined depending on the type of device as well as its specifications during the initial setup. This ensures optimal detection of anomalies while maintaining a stable process with the capabilities which are otherwise typical in the IoT network with an emphasis on device constraints. In case of a system reconfigure, the threshold values are computed accordingly.

This expression defines the prediction criteria based on T_{CPU} and distinguishes between normal and potentially malicious activity. When the CPU usage is below T_{CPU}, the system is operating under normal operations. However, if the CPU usage meets or exceeds T_{CPU}, the node is under attack. This approach, although simple,

can serve as an effective initial layer of defence in detecting attacks based on resource utilization patterns.

$$\text{Prediction} = \begin{cases} \text{Normal} & \text{if CPU Usage} < T_{\text{CPU}} \\ \text{Attack} & \text{if CPU Usage} \geq T_{\text{CPU}} \end{cases}$$

Memory: The threshold value for memory usage is established by averaging the Max memory usage recorded during normal operational conditions and the Min memory usage observed during an attack. The threshold is calculated using the following Equation (5.2):

$$T_{\text{Memory}} = \frac{\max(\text{Memory}_{\text{Normal}}) + \min(\text{Memory}_{\text{Attack}})}{2} \tag{5.2}$$

After applying the given equation, the memory threshold (T_{Memory}) is established by averaging the peak memory usage under normal conditions and the lowest memory consumption observed during an attack. In this case, the peak normal utilization of memory is 48.3 MB, and the minimum utilization of attack memory is 37.9 MB, as shown in Figure 5.4. It leads to a calculated threshold value of 43 MB. The threshold value provides a point of comparison for separating normal and possibly malicious behaviors with respect to memory utilization patterns. The prediction can be stated as follows:

$$\text{Prediction} = \begin{cases} \text{Normal} & \text{if Memory Usage} < T_{\text{Memory}} \\ \text{Attack} & \text{if Memory Usage} \geq T_{\text{Memory}} \end{cases}$$

Packet rate: Formula (5.3) calculates the average of the highest packet rate during normal operation and the lowest packet rate during attack phases in order to set a threshold for the rate at which packets are received by an IoT device. It serves as a comparison yardstick for separating normal and possibly malicious traffic based on packet rate behaviors.

$$T_{\text{Packet\_rate}} = \frac{\max(\text{Packets}_{\text{Normal}}) + \min(\text{Packets}_{\text{Attack}})}{2} \tag{5.3}$$

In this particular case, the smallest attack packet rate value corresponds to 19,681 packets per second, and the largest normal packet rate value equals 4 packets per second. By calculating the arithmetic mean of these two values, a threshold value for packet rate, denoted as $T_{\text{Packet\_rate}}$, is defined as 9,842.5 packets per second. This particular value serves as a baseline for comparison intended to distinguish normal and possibly malicious network behaviors based on packet rate patterns. The prediction can be expressed as follows:

$$\text{Prediction} = \begin{cases} \text{Normal} & \text{if Packet Rate} < T_{\text{Packet\_rate}} \\ \text{Attack} & \text{if Packet Rate} \geq T_{\text{Packet\_rate}} \end{cases}$$

5.3.2 Health Monitoring

The thresholds serve as important markers to measure the health status of IoT devices. The health status tracking module continuously monitors resource usage, and if it exceeds its respective thresholds, a danger signal is triggered as indicated in Algorithm 5.1. The algorithm utilizes the psutil library to determine the current usage

of CPU, memory, and packet rate and compares them with predefined thresholds indicated by τ in Algorithm 5.1. Moreover, it sets a counter for the signal_number to zero and uses an iterative loop to continuously monitor resource usage. In each iteration of the loop, it retrieves the current usage of CPU, memory, and packet rate resources. In the event the usage exceeds its respective thresholds, it generates a danger signal with the IP address of the device, the current signal_number, and the timestamp. The danger signal is crucial in facilitating an effective response to threats. Also, the danger signal is triggered only when the usage of resources goes beyond the set thresholds, thus ruling out the possibility of an incorrect warning due to normal device operations. Upon generation, the danger signal is sent to the IoT gateway for processing. Until the healing process is completed, the IoT node sends out the kill signal to any process utilizing CPU usage greater than the set threshold (41.7%).

5.3.3 Packet Collection

Step II consists of three modules, as depicted in Figure 5.1, each executing specialized instructions tailored to specific tasks. This phase, taking place at the network layer, initiates when a danger signal is triggered by the victim IoT node due to resource usage surpassing predefined threshold values.

Algorithm 5 .1: Resource Monitoring and Danger Signal

Input: Threshold (τ) values for CPU, memory, and packet_rate.

Output: Generate a danger signal: *IP_address*, *signal_number*, *timestamp*.

1: Initialize the *signal_number* to zero.
2: **while** true **do**
3: Retrieve the current values for $\text{CPU}_{\text{usage}}$, $\text{memory}_{\text{usage}}$, and packet_rate.
4: **if** $\text{CPU}_{\text{usage}} > \tau_{\text{CPU}}$, $\text{memory}_{\text{usage}} > \tau_{\text{memory}}$, and packet_rate $> \tau_{\text{packet\_rate}}$ **then**
5: Generate a danger signal encompassing (*IP_address*, *signal_number*, *timestamp*), signal_number = signal_number + 1
6: Retrieve the process ID of the process with CPU usage exceeding the threshold τ_{CPU}, and send a kill signal.
7: **end if**
8: **end while**

On receipt of the danger signal, the packet sniffing module follows the set of actions described in Algorithm 5.2. The actions outlined in the algorithm obtain primary details such as the IP address of the victim node, the signal number, and the timestamp of the danger signal. The danger signal timestamp is important since it helps in attack timing tracking. Moreover, the module triggers the packet sniffer through the execution of "tcpdump.exe" with Flag = 1. The flag identifies that the self-healing procedure has started.

Following Zhang et al. (2024)'s recommendation [178] that a minimum of 50,000 80,000 samples be used for retraining, this will help ensure improved differentiation between malicious traffic and legitimate network traffic. As such, packet

sniffing will be set to stop at 74,208 packets since this reflects the number of packets that pass through the network traffic during the simulation of a DoS/DDoS attack, explained in section 5.3.

Algorithm 5 .2: Receiving Danger Signal and Initiate Packet Capturing

Input: Danger_signal (*IP_address*, *signal_number*, *timestamp*)

Output: Signal to tcpdump, set value Flag to 1

1: Start
2: Input: Danger_signal
3: Extract deviceID ← Danger_signal, signal_number ← Danger_signal, timestamp ← Danger_signal
4: open(logFile, appendMode)
5: write [logFile ← (device_ID, timestamp, signal_number)]
6: close(logFile)
7: Signal tcpdump to start packet capturing, and Set Flag to 1
8: End

With regard to the IoT node that is under attack, the average packet rate that is observed is 24,736 pps. This packet rate strikes a balance between the computational complexity and the packet density that is sufficient for binary classification. It captures 74,208 packets so that a representative sample of the packet flow that includes attack as well as normal traffic is obtained.

Also, gathering 74,208 packet samples at 24,736 pps gives an estimated capture duration of about 3 seconds, thus capturing an optimal trade-off between reactivity and the comprehensiveness of capture duration. Such capture duration is highly relevant for facilitating the immediate execution of self-healing, thereby enabling

rapid detection and an instant response before mounting further attacks. A short capture duration guarantees an appropriate amount for analysis without overloading the resources on the network.

The packets are also useful not only for the real-time detection of attacks but also for the optimization of the online learning process for the intrusion detection system (IDS model). The number of packets, which is 74,208, is sufficient to provide a good amount of data for model updating to enable it to be able to discriminate between the normal and attacking data during the course of the attack. The packets collected are considered data for learning that the system uses to improve the model's efficiency to detect the attacks.

Furthermore, the system is designed with resource effectiveness in mind, with the packet capture limited to this predetermined number, thus overcoming potential performance hindrances arising from excessive overhead (as identified in [179]. The system's ability to respond promptly is important for ensuring the integrity of IoT devices as well as network security. Even if 74,208 packets may be considered inadequate for capturing high-volume attacks, the system is flexible enough to respond to various attack types. For example, in cases where low-and-slow DDoS attacks or lower transmission rates are involved, the system would be capable of switching between packet capture sessions depending on different attack phases. With the completion of the packet capture, the captured data is stored in a PCAP format, as demonstrated in Algorithm 5.3. The use of the PCAP format ensures that all the necessary details are taken into account, such as the headers and payloads of the packets, making it suitable for network analysis. For easy manipulation and analysis

of the data, the PCAP file is converted to a CSV format. This process begins by opening the target PCAP file to read the data and outlining a new CSV file to write the derived attributes. The conversion of the file to CSV format makes it easy to execute the steps of clustering and machine learning model retraining. The packets are then passed to the application layer.

5.3.4 Retraining

In this research, resource-consuming modules are placed in the fog level in order to execute complex tasks in an efficient manner without stressing IoT devices. The labelled samples of data are created using a network packet file that is given as an input to the K-means algorithm, which is an unsupervised machine learning technique mostly used in clustering and assigning labels to unlabelled data. K-means is well-known for its processing speed in computation, particularly in working with massive amounts of data or in a real-time setting [180]. It also produces accurate clusters from the input data, ultimately enhancing analytical outcomes. In this study, the parameters and their detailed configuration are detailed in Table 5.2.

The number of clusters to be created is indicated by the parameter n_clusters within the K-means clustering algorithm. For the experiment, n_clusters is set to 2, where the first cluster targets normal traffic data while the second targets the attack data. In order to ensure that the cluster labeled normal actually contains normal traffic data, the packets are cross-validated based upon the times of danger signal occurrences, where packets detected outside the times of generation are considered normal packets while others target attacks.

Algorithm 5 .3: Convert PCAP File to CSV with All Packet Attributes

Input: pcapFile (PCAP file containing captured network traffic)

Output: csvFile (CSV file with extracted packet attributes)

1: Start

2: Open pcapFile for reading.

3: Create csvFile and open it for writing.

4: Write the CSV header to csvFile:

5: header ← ["Timestamp", "Source MAC", "Destination MAC", "Ethernet Type", "Source IP", "Destination IP", "Source Port", "Destination Port", "Protocol", "Packet Length", "TTL", "IP Flags", "Fragment Offset", "TCP Flags", "TCP Window Size", "Sequence Number", "Acknowledgment Number", "UDP Length", "ICMP Type", "ICMP Code", "Payload Length"]

6: write (CSV, header)

7: **for** each packet P in pcapFile **do** Extract the attributes from packet P:

Time, S_MAC, D_MAC, Eth_Type, S_IP, D_IP, TTL, IP_Flags, Frag_Offset, Proto, S_Port, D_Port, Seq_Num, Ack_Num, TCP_Flags, TCP_Window, UDP_L, ICMP_Type, ICMP_Code, L, Payload_L, timestamp, source_mac, destination_mac, ethernet_type, source_ip, destination_ip, ttl, ip_flags, fragment_offset, protocol, source_port, destination_port, sequence_number, ack_number, tcp_flags, tcp_window_size, udp_length, icmp_type, icmp_code, packet_length, payload_length

8: Write a CSV record:

9: row ← [Time, S_MAC, D_MAC, Eth_Type, S_IP, D_IP, S_Port, D_Port, Proto, L, TTL, IP_Flags, Frag_Offset, TCP_Flags, TCP_Window, Seq_Num, Ack_Num, UDP_L, ICMP_Type, ICMP_Code, Payload_L]

10: **end for**

11: End

Table 5.2: Configuration parameters for K-means clustering

Parameter	**Value**
n_clusters	2
init	K-means++
max_iter	300
n_init	10
algorithm	elkan

In order to enhance the initialization of clustering, the K-means++ algorithm is employed for the selection of the initial cluster centers [181]. Unlike random selection for the initialization of clustering, which can cause adverse convergence and could be trapped in a local optimum, the selection of the initial cluster centers for clustering can be done using K-means++ with an increased possibility of achieving a beneficial clustering partition [182].

A maximum of 300 iterations within a single run is allowed to ensure convergence of the algorithm. Convergence of the algorithm occurs when the centroids no longer experience drastic changes in values [183]. Setting a limit to the iterations is important to prevent the algorithm from running indefinitely, most specifically in handling large and complex datasets. Although a maximum of 300 iterations is generally appropriate for most datasets, this can be varied depending upon the characteristics of the datasets [183].

Further, the n_init parameter is used to determine the number of unique initial

centroid configurations that will be assessed before determining the final solution. In this analysis, n_init is defined as 10; this means that the K-means algorithm will run 10 times with randomly selected centroid configurations. These steps help overcome local optima issues since a variety of initial points are tried, and the centroid configuration with the lowest sum of squares of errors is chosen. A higher n_init is considered more favorable in obtaining a globally optimal solution but is associated with higher computation time.

Lastly, the algorithm parameter specifies the appropriate algorithm to apply when performing K-means clustering. There are three choices: auto, elkan, or full.

- **Auto:** This algorithm chooses the best method based solely on the data.
- **Elkan:** The Elkan algorithm can be regarded as an optimized version of the K-Means algorithm that uses the triangle inequality in order to speed up the computation by minimizing the number of calculations of distances [184]. The Elkan algorithm works best when the data is dense.
- **Full:** The full algorithm calculates the distance from each data point to all centroids at each step, thus making computation expensive but capable of working with all kinds of data.

For this purpose, the Elkan algorithm is chosen to conduct the experiment because of its efficiency and faster execution speed as compared to the traditional algorithm. The Elkan method proves to be highly efficient in case of dealing with a large number of features or when there are certain constraints in terms of computational

capabilities [185]. In such a situation, Elkan helps in overcoming the problem of high time complexity associated with the K-means clustering method [185].

The dataset with samples marked is then fed into the machine learning (ML) algorithm for retraining the model. After retraining and evaluation, the model achieves good performance (above 95% accuracy) and a FPR of 0. The retrained model is then coupled with the pre-existing model. In this coupling, a reset signal is sent to the "Packet Sniffing" module, and Flag is reset to 0. The reset signal acts as an important notification that indicates the self-healing operation has successfully completed and is ready for responding to threat notifications in the future. With the inclusion of the retrained model, the system benefits from gaining the most up-to-date knowledge gleaned from retraining while retaining its core knowledge from the pre-existing model. In this regard, the Alert Receiver module is ready for monitoring anomalies and responding as needed. After this stage, the performance of the retrained Intrusion Detection System (IDS) model will be tested in subsequent stages.

5.3.5 Performance Evaluation Parameters for Proposed SH-IoT

After the training process, the performance and resource use of the resulting models are analyzed. This paper evaluates the models based on their accuracy, training time, and test time. Moreover, CPU and Memory usage are also analyzed during the execution of the models. As indicated by [160] during the process of benchmarking, each binary classification model provides at least four key measures of its perfor-

mance based on the confusion matrix results, such as True Positive Rate (TPR), also called sensitivity and recall, True Negative Rate (TNR), also called specificity, False Positive Rate (FPR), and False Negative Rate (FNR). These measures apply for the machine learning models proposed for the intrusion detection system (IDS) as discussed in Chapter 3.

Training Time: This is the time required to complete various sub-tasks before the commencement of training the model. In the self-healing process, training time includes data preparation, clustering techniques (K-means), as well as the time required to retrain machine learning models as given in Equation (5.4).

$$\text{Total Training Time} = T_{\text{preprocessing}} + T_{\text{clustering}} + T_{\text{training}} \tag{5.4}$$

where $\text{Time}_{\text{Clustering}}$: Time required for labeling samples using K-means clustering, $\text{Time}_{\text{Preprocessing}}$: Time required for data preprocessing tasks and $\text{Time}_{\text{Training}}$: Time required during the training of ML classifiers.

Testing Time: Testing time is the time taken by the trained machine learning classifier to process instances of input data to produce a prediction. Testing time is important in scenarios where a real-time prediction is required. Testing time is calculated using the formula shown in Equation (5.5).

$$\text{Testing Time} = \text{End\_Time}_{\text{test}} - \text{Start\_Time}_{\text{test}} \tag{5.5}$$

where $\text{Start\_Time}_{\text{test}}$ is the time recorded just before the testing or prediction process begins and $\text{End\_ Time}_{\text{test}}$ is the time recorded immediately after the testing

or prediction process ends.

For the measurement of the training period and testing period of the classifier, the time module is used in Python due to its ability to measure the execution time of the code in milliseconds. However, it is important to note that the measurement may vary depending on the parameters such as hardware requirements, complexity, and machine learning parameters.

Memory Utilization During Training: A memory utilization implemented during the training stage refers to the amount of computer memory used for activities such as processing datasets, grouping, as well as developing machine learning models. The memory consumption during this stage is calculated using Equation (5.6).

$$\text{Total Memory}_{\text{train}} = \text{Memory}_{\text{Clustering}} + \text{Memory}_{\text{Preprocessing}} + \text{Memory}_{\text{Training}} \quad (5.6)$$

where $\text{Memory}_{\text{Clustering}}$ memory required for labeling samples using clustering, $\text{Memory}_{\text{Preprocessing}}$ memory required for data preprocessing tasks and $\text{Memory}_{\text{Training}}$ memory required during the training of ML classifiers.

In this way, the equation provides a comprehensive view of the memory required for each component involved in the training phase of the machine learning model.

Memory Utilization During Testing: In the testing phase, memory is utilized for processing input features, loading the trained model, and prediction results

during the application of a trained model to test data samples is given in Equation (5.7).

$$\text{Memory}_{\text{test}} = \text{Memory}_{a_{\text{test}}} - \text{Memory}_{b_{\text{test}}} \tag{5.7}$$

where $\text{Memory}_{a_{\text{test}}}$ is the memory usage measured before the testing process starts and $\text{Memory}_{b_{\text{test}}}$ is the memory usage measured immediately after the testing process ends.

CPU Usage During Training: During the training phase, the cumulative CPU requirements for all the processes involved can be identified clearly. This includes CPU requirements for processing the input data, updating the centroids as well as the processing involved during the clustering operation. The data preprocessing operation utilizes the CPU for all the cleansing and processing involved for the purpose of preparing the data for the machine learning model training phase. These cumulative CPU requirements can be calculated as indicated in Equation (5.8).

$$\text{Total CPU}_{\text{train}} = \text{CPU}_{\text{Clustering}} + \text{CPU}_{\text{Preprocessing}} + \text{CPU}_{\text{Training}} \tag{5.8}$$

where $\mathbf{CPU}_{\text{Clustering}}$ is the CPU utilization during labeling, $\mathbf{CPU}_{\text{Preprocessing}}$ CPU required for data preprocessing tasks and $\mathbf{CPU}_{\text{Training}}$ CPU needed during the training of ML classifiers.

CPU Usage During Testing: In the testing phase, the CPU is used to process input features, load the trained model, and predict results during the application

of a trained model to test data samples are given in Equation (5.9).

$$\mathrm{CPU}_{\mathrm{test}} = \mathrm{CPU}_{a_{\mathrm{test}}} - \mathrm{CPU}_{b_{\mathrm{test}}} \tag{5.9}$$

where $\mathrm{CPU}_{a_{\mathrm{test}}}$ is the CPU usage measured before the testing process starts and $\mathrm{CPU}_{b_{\mathrm{test}}}$ is the CPU usage measured immediately after the testing process ends.

The current study leverages the psutil library for resource monitoring during the model development stage of machine learning as well as during the testing phase. The psutil library is an open-source library that was developed to obtain information on resource usage.

5.4 Summary

The self-healing technique treated in this chapter is related to health monitoring of IoT nodes based on resource usage analysis under normal conditions and DoS/DDoS attacks. Under a normal scenario, an IoT node handles around 4-5 packets per second, signifying effective usage of resources. During an attack, this number may rise to several thousand per second, significantly burdening the CPU and memory of the IoT node in question. In a bid to avert a possible failure in an IoT node due to occurrences of DoS/DDoS attacks, a self-healing system containing health monitoring, packet gathering, clustering, and retraining of models has been proposed.

IoT systems are characterized by being heterogeneous systems; however, the experimental environment used in this work represents a homogeneous environment. However, the self-healing mechanism developed in this work is not only limited to a

homogeneous environment. The mechanism can also be applied to a heterogeneous IoT environment by incorporating device configurations with specific threshold values for each device type.

Looking ahead, the threshold values used in the proposed self-healing process will take into account the adaptive thresholds that change in real-time according to network dynamics and trends. Additionally, a process will also be developed that will enable automatic isolation of the devices from the IoT network according to the number of warnings they produce. Based on signal_number, which is used to track warning signals, devices that produce anomalies will be identified. The findings of the experimentations related to effectiveness of the self-healing process in improving adaptability and performance of IDS will be reported in the next chapter.

CHAPTER 6

Performance Evaluation of SHIoT Mechanism for ML-Enabled IDS

6.1 Introduction

This chapter presents the comprehensive experimental results of the proposed SH-IoT mechanism proposed in Chapter 5. The discussion covers the experiments and results obtained using the BoT-IoT dataset. Additionally, the retrained model is further evaluated using the TON_IoT dataset to assess the adaptability of the retrained models with TON_IoT dataset. The evaluation includes graphs for accuracy, confusion matrix parameters, retraining and testing time, memory usage, and CPU usage.

As illustrated in Figure 6.1, the BoT-IoT data samples are used to evaluate the self-healing mechanism. This dataset is available in both CSV and PCAP formats.

In this chapter, we use the data samples from the PCAP file to properly execute each step in the self-healing mechanism. Furthermore, the unlabeled samples are input into K-means clustering. After clustering, the labeled samples are split into training and testing sets. The training set is used to retrain the ML models (RF, DT, GB, KNN, NB, and NN), which were previously trained on the TON_IoT dataset. Finally, the updated models are evaluated based on various performance metrics, including accuracy, training and testing time, CPU and memory usage, and confusion matrix parameters. This self-healing mechanism ensures that the IDS can effectively detect and adapt to new DoD/DDoS attacks.

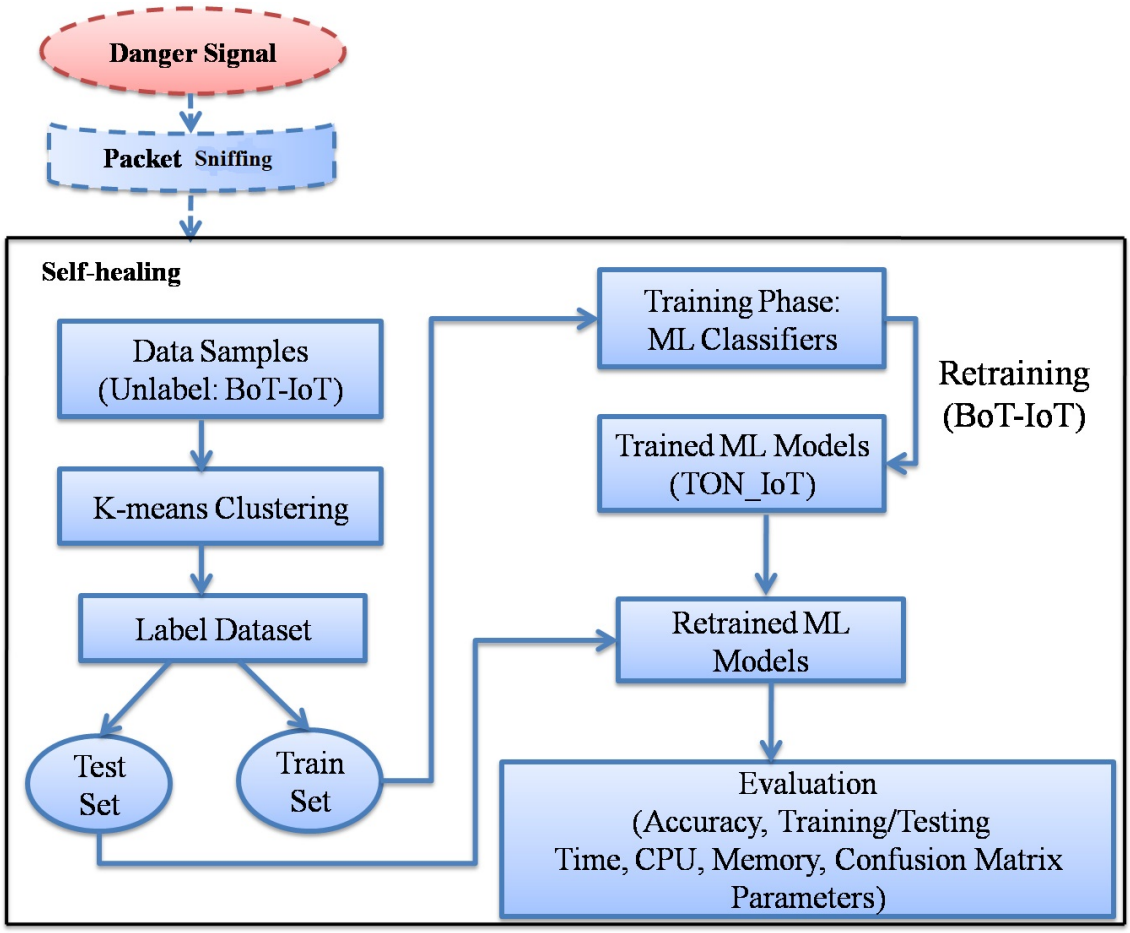


Figure 6.1: Performance Evaluation Process of the Self-Healing Mechanism.

6.2 Self-healing Evaluation and Discussion

This section presents the experiments validating the self-healing mechanism and provides insights into the results obtained. The section comprises six parts: Experimental setup, Health monitor and danger Signal, clustering and labeling, model retraining, results and analysis based upon BoT-IoT test samples, and model testing based on ToN_IoT test samples, as detailed in subsequent sub-sections.

6.2.1 Experimental Setup for the Self-Healing Mechanism

In this Chapter, all experiments are performed using the same tools, classifiers' configuration, and techniques used in Chapter 4. For example, Scikit learns v_0.24.2 libraries, Pandas v_1.3.4, and NumPy v_1.20.3 under the distribution of anaconda with Python version 3.9.7. Configuration for the computer system utilized for executing the experiments includes a Core (TM) i5-6600 CPU @ 3.30GHz processor, 8 GB memory, where the system is running Windows_10 Pro OS. Furthermore, the Python "psutil" library is utilized to monitor resource usage, which includes the monitoring of CPU utilization, memory consumption, and training and testing time required with each learning algorithm.

In this experiment, we use the BoT-IoT dataset to extend the analysis presented in Chapter 4, where IDS models trained on the TON_IoT dataset and evaluated on the BoT-IoT dataset showed a decline in performance during cross-domain evaluation, as illustrated in Figure 4.10. To enhance attack detection and strengthen the IDS's robustness, we continue the evaluation using the BoT-IoT dataset.

6.2.2 Health Monitor and Danger Signal

The self-healing process is triggered by the generation of the danger signal itself, denoting the deterioration of the health status of an IoT device below specific threshold values. Throughout the experimental analysis, we kept track of the IoT device's performance parameters and thereby deduced the correlation between the number of DoS/DDoS attack packets and the health monitor values, both necessary for the generation of the danger signal. As shown in Figure 6.2, the health monitor generates the danger signal once the memory usage surpasses 43MB. Before that happens, the system does not produce the danger signal despite satisfying other conditions such as having a packet rate above 9,843 pps and CPU usage above 41.7%. This indicates that the system requires all three predefined threshold levels to be met on all resources at once for the danger signal generation. Such an approach is highly necessary since an increased packet rate may not always indicate an improbable event but may also be due to system factors such as its concurrent effect and environment settings that turn out not to pose threats at all. As shown in Figure 6.2, the danger signal is triggered at 20.4 ms, denoting the point at which all resources exceed their corresponding threshold levels.

With the integration of a health monitor that takes into consideration memory usage, CPU usage, and packet rate, the system is not easily vulnerable to unnecessary notifications and thus improves robustness and prevents false indicators of danger. The process ensures that packet rates are not taken out of context and that the process of self-healing is not prematurely triggered. The system is thus better

able to respond to actual threats while conserving and maintaining stability.

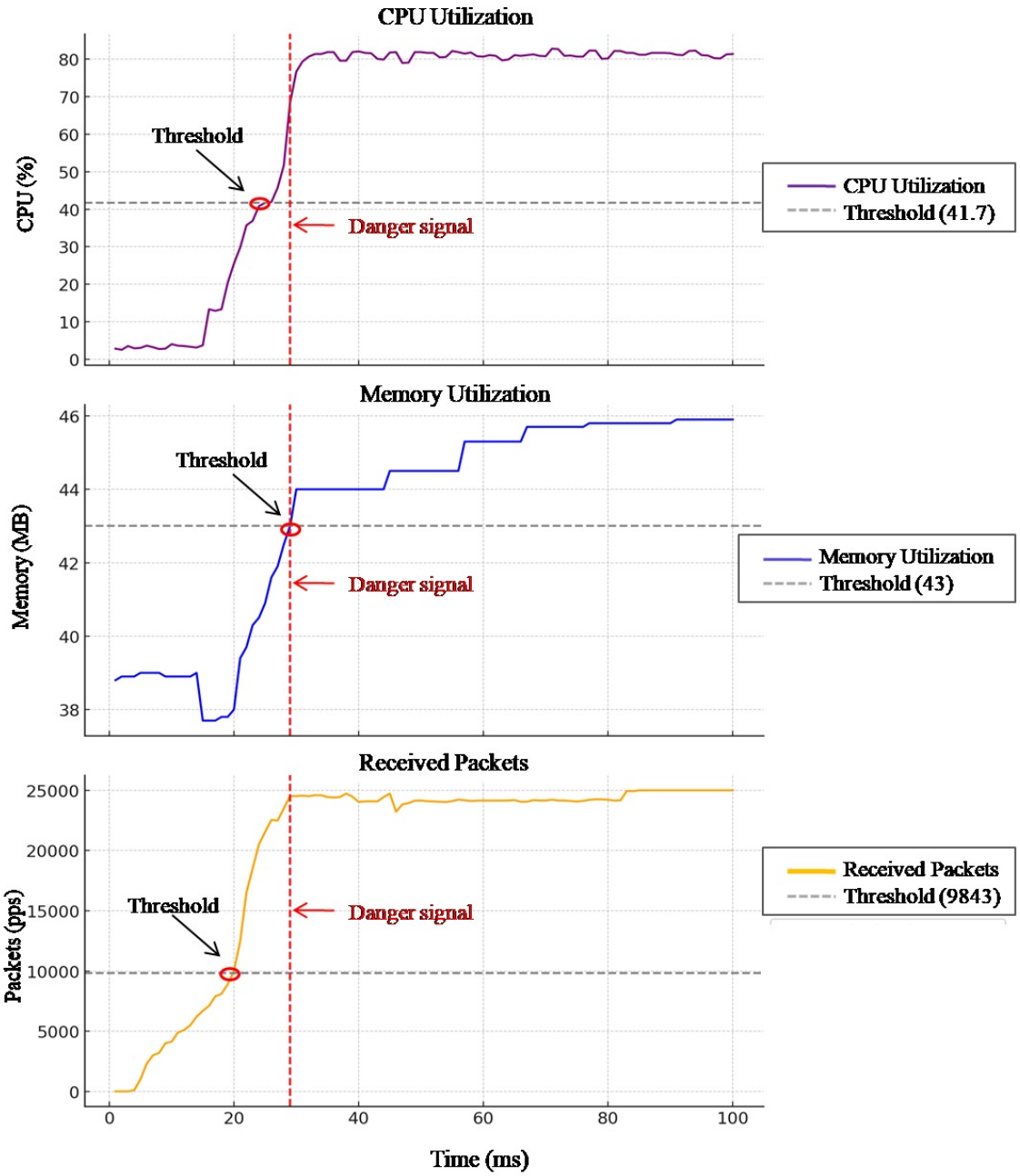


Figure 6.2: Alert for the danger signal

6.2.3 Clustering and Labeling

To check the accuracy of K-means clustering in a systematic way, PCAP files containing both malicious and normal traffic data of the publicly available BoT-IoT

dataset are used. From these PCAP files, a dataset of 80,000 records is formed, equally split into 40,000 normal network traffic records and 40,000 DoS/DDoS attack records. The PCAP files are then converted to CSV format, retaining 38 features that serve as variables to the K-means clustering algorithm. K-means clustering, an unsupervised machine learning algorithm, is then used to create the clusters as in Figure 6.3.

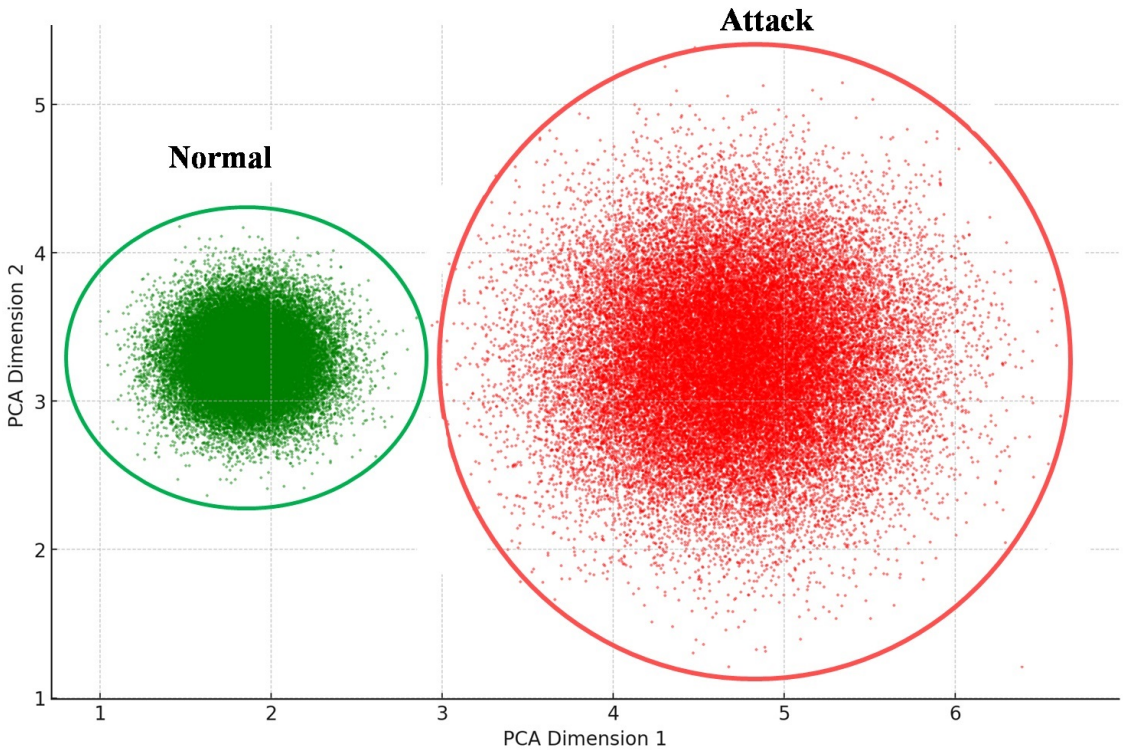


Figure 6.3: Normal and attack clusters

Typical implementation after clustering involves labeling, which is carried out based on the usage of timestamp data together with the K-means algorithm. The danger signal timestamp is an important point of reference in correlating packets with attack periods. For K-means, although data points can be separated into two clusters based on features, these clusters are not labeled. To label these clusters with

important features, they have to be validated using danger signal timestamps. Packets during active danger periods are mostly attack packets; thus, the bigger cluster containing correspondences with the danger time period is labeled 'Attack,' and the other is 'Normal traffic.' In this manner, the clusters will definitely correspond with actual realities in operation.

The resulting dataset obtained through clustering contains 39 attributes of these packets, along with an extra one called Label that classifies each observation into either the "Normal" category or "Attack" class. This labeled dataset is of prime importance to the machine learning classifiers for their training process of developing an intrusion detection system model. This approach of using both time-domain information through clustering and ensuring that a labeled dataset is prepared helps to achieve accurate automatic labeling that is more useful to address large datasets with their complexities of manual labeling.

6.2.4 Model Retraining

Self-healing by means of retraining is one of the primary methods aimed at creating resilient ML-based intrusion detection systems. The method focuses on the continuous improvement of models by feeding them new data. The retraining cycle involves updating ML-based IDS models periodically by feeding them new instances. The continuous learning process enables the models to be efficient in identifying anomalies and adjust to the realities of practical network traffic. In the end, retraining is an aid in the creation of an efficient framework that provides un-interrupted functionality by handling resources and reacting to potential threats; in total, it enhances

the security posture of IoT networks by identifying both current and novel attacks.

The classifiers RF, DT, GB, KNN, NB, and NN are re-trained on the samples with the machine learning models. Their parametric settings are kept consistent with the descriptions presented in Chapter 4. Before the re-training, there is pre-processing on the data with the aim of allowing effective learning with the machine learning tools. During the preprocessing, the use of label encoding helps in the encoding of the categorical variables, thus enabling the conversion of the non-numeric variable into their suitable format. Furthermore, during the preprocessing, the normalizing/preprocessing stage involving Min-Max scaling is carried out to scale the variables appropriately, thus avoiding the dominance on the characteristics during the designing due to their differences brought about by their magnitudes.

After preprocessing, 10-fold cross-validation is used for splitting the dataset into training and testing sets. For this analysis, each fold contains 80,000 instances and 39 features, where 72,000 instances are used for training and 8,000 samples are used for testing. This stratified process for testing is used where the model is tested on different samples, increasing the robustness of testing accuracy. For training samples, six classifiers from machine learning algorithms, namely RF, DT, KNN, GB, NB, and NN, are used independently for their respective model training with parameters listed in detail in Chapter 4 in Table 4.6. These trained models are then tested independently using 8,000 test samples. Results and corresponding plots will be given in the succeeding section.

6.2.5 Performance Evaluation using BoT-IoT Dataset

This section gives an in-depth analysis of the results achieved on the BoT-IoT dataset by applying different ML algorithms to identify an attack in the IoT network flow. Figures 6.4 to 6.11 show the different evaluation criteria, which include accuracy, TPR, FNR, TNR, and FPR. As resources in the IoT are limited, the analysis will cover the usage and training/test time of the ML algorithms.

The accuracy shown in Figure 6.4 highlights a dramatic improvement in the performance of the ML models after retraining. Notably, this is especially true when contrasted with the accuracy shown in Figure 4.9 from Chapter 4, wherein ML models were trained using the TON_IoT dataset and then tested using the BoT-IoT dataset in a cross-domain scenario. The retraining phase allowed the classifiers to learn better from and adjust themselves according to the unique patterns in the BoT-IoT dataset, thus resulting in higher accuracy values. For instance, the accuracy of the classifiers increased significantly for RF, GB, and DT models from 69.8%, 58%, and 50%, respectively, to values close to 99.9%. The accuracy for the NN also improved from 53.6% to 95%.

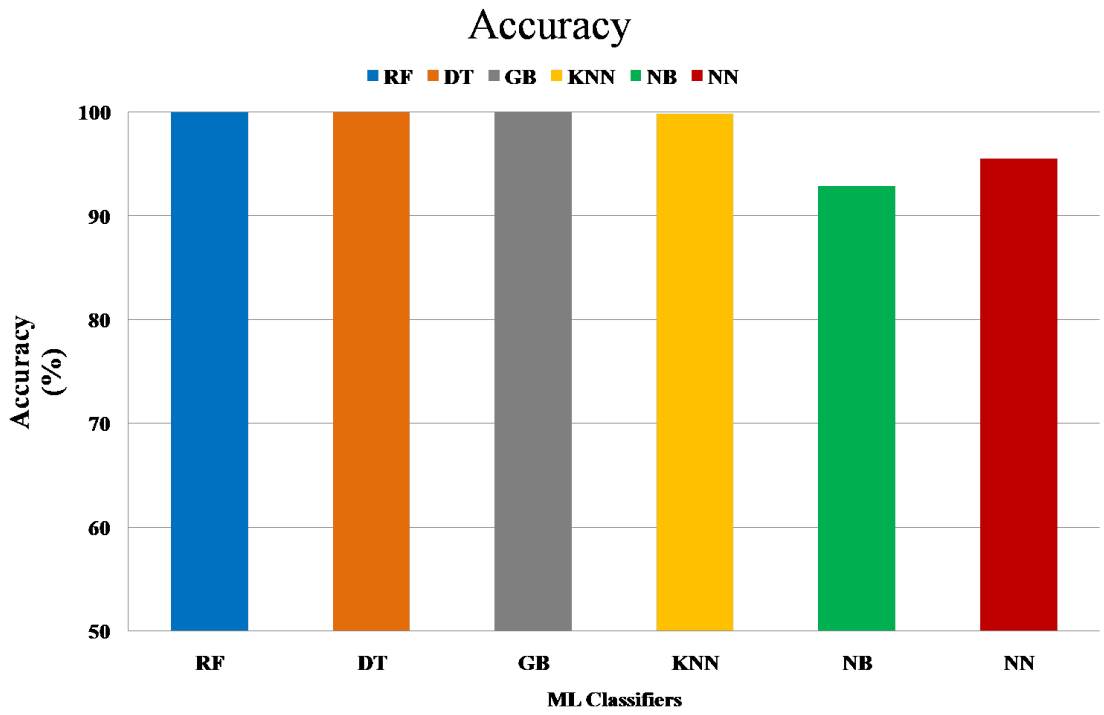


Figure 6.4: Accuracy after self-healing using BoT-IoT dataset

The performance metrics in Figures 6.5 (a, b, c, and d) show the TPR, TNR, FNR, and FPR of various machine learning classifiers. According to Figure 6.5(a), TPR equals 1 for all four ML classifiers: RF, DT, k-NN, and GB, meaning that all the attack examples are perfectly classified by these ML algorithms. On the contrary, in Figure 6.5c, FNR is portrayed, of which NN possesses an FNR of 0.1 and NB possesses an FNR of 0.01. On the other hand, RF, DT, KNN, and GB did not classify any DoS/DDoS examples as 'Normal' examples. Likewise, in Figure 6.5b, TNRs of 1 are demonstrated for RF, DT, KNN, and GB, whereas TNRs of 0.9 were demonstrated by NB and NN classifiers. Lastly, Figure 6.5d, it is demonstrated that NB and NN possess the highest FPR of 0.02, meaning that the TPR of the other ML algorithms is perfect and did not classify 'Normal' examples as 'Attack'.

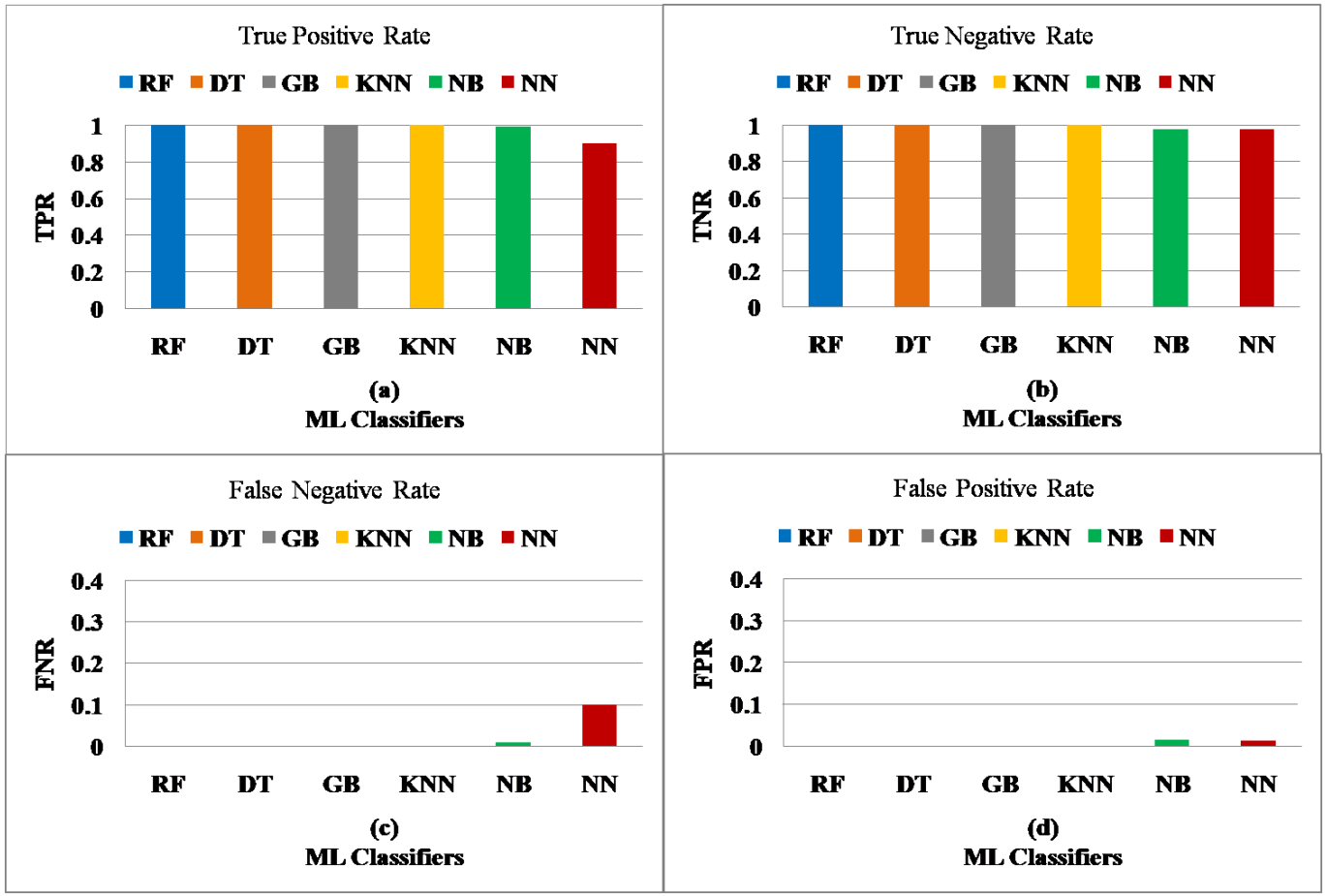


Figure 6.5: Performance metrics using BoT-IoT dataset (a) TPR (b) TNR (c) FNR and (d) FPR

From the above figure, it is clear that RF, DT, KNN, GB classifiers perform better in all the evaluated metrics to classify the attack and normal traffic. The experiment considers time and overhead in training and testing to analyze the efficiency of binary classification models based on machine learning algorithms on the BoT-IoT dataset. Figure 6.6 shows the training time of six machine learning classifiers that work on 39 features in BoT-IoT.

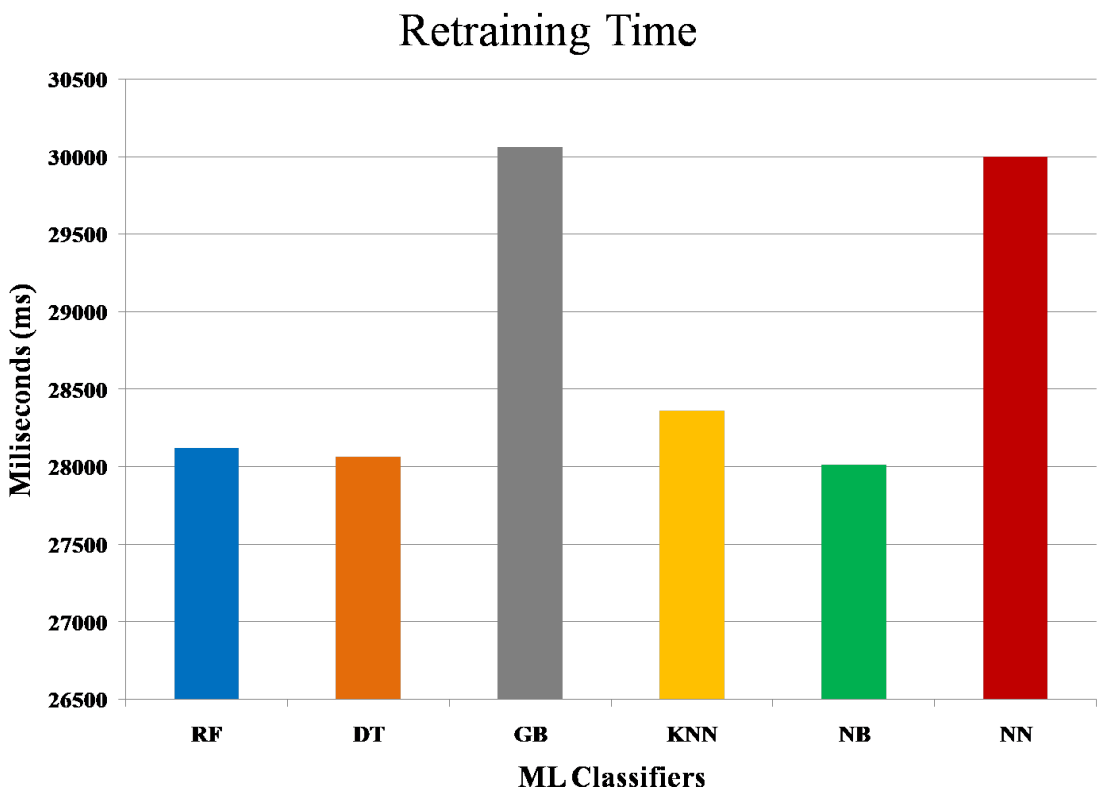


Figure 6.6: Time required to retrain ML classifiers

Analysis of the output reveals that Naive Bayes has a lower retraining time of 28,013.3 ms compared to other classifiers. In fact, even Decision Tree and Random Forest take slightly higher retraining times of 28,063.9 ms and 28,120 ms, respectively. However, Gradient Boosting and Neural Network take the highest retraining times of approximately 30,060 ms and 30,000 ms, respectively. The high retrain-

ing times for Gradient Boosting and Neural Network can be attributed to the fact that they are complicated models and resource-intensive. Gradient Boosting is an ensemble learning model that uses various weak models to generate a strong predictive model [186]. In addition to that, Neural Network uses a multi-layered neural network that uses a backpropagation concept [187].

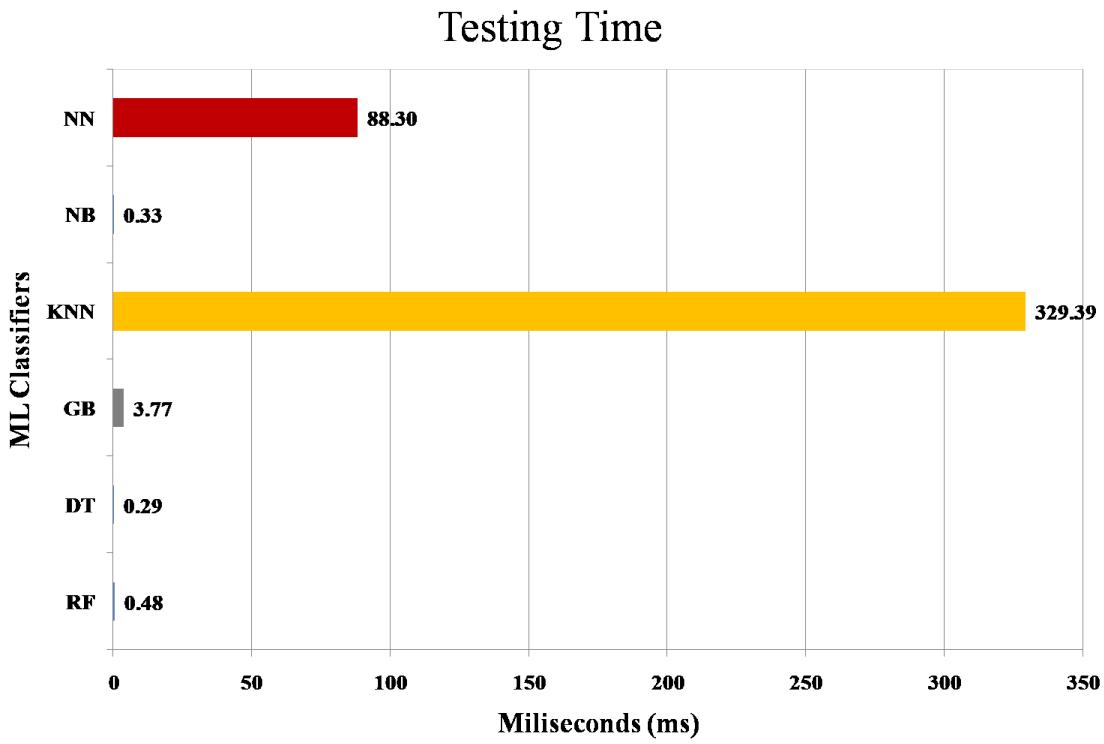


Figure 6.7: Per sample testing time after self-healing

Furthermore, Figure 6.7 shows the testing time required to test BoT-IoT instances individually. From the results, it is observed that DT takes the lowest testing time among the classifiers with a testing time of 0.29 ms. However, NB and RF take slightly higher testing times of 0.33 ms and 0.48 ms, respectively. On the other hand, KNN takes the highest testing time of 329.39 ms due to the distance calculations between the testing sample and the training samples.

Moreover, Figure 6.8 also explains the memory consumption of retraining different machine learning classifiers, such as RF, DT, GB, KNN, NB, and NN. It is clear that NB consumes less memory, which is only 2264 MB, indicating its relatively simple probabilistic model that requires less computational memory. The second least memory consumers are RF and DT, which take slightly higher memory of 2280 MB and 2278 MB, respectively. Moderate memory consumption is attributed to the simplicity of the tree structure of DT and the use of a small number of trees in RF for ensemble learning. On the contrary, NN consumes the highest memory of 2539 MB. The second-highest memory consumers are GB and KNN.

Figure 6.9 presents the memory usage for every trained ML model during the testing phase. The experiment results show that every model, except KNN, uses a moderate level of memory, while KNN uses the most memory, up to 4.96 MB, because it stores the entire training data to calculate the distances during the testing phase, thereby using a considerable amount of memory. The minimum memory usage, less than 4 MB, is shown by RF, DT, GB, and NB models during the testing phase to classify a BoT-IoT instance. The memory usage of 4.39 MB occurs in the NN model because it is a multi-layer model, retaining every learned operation, including the weights and activations, during every layer to make accurate predictions, thus using a considerable amount of memory compared to RF, GB, DT, and NB models.

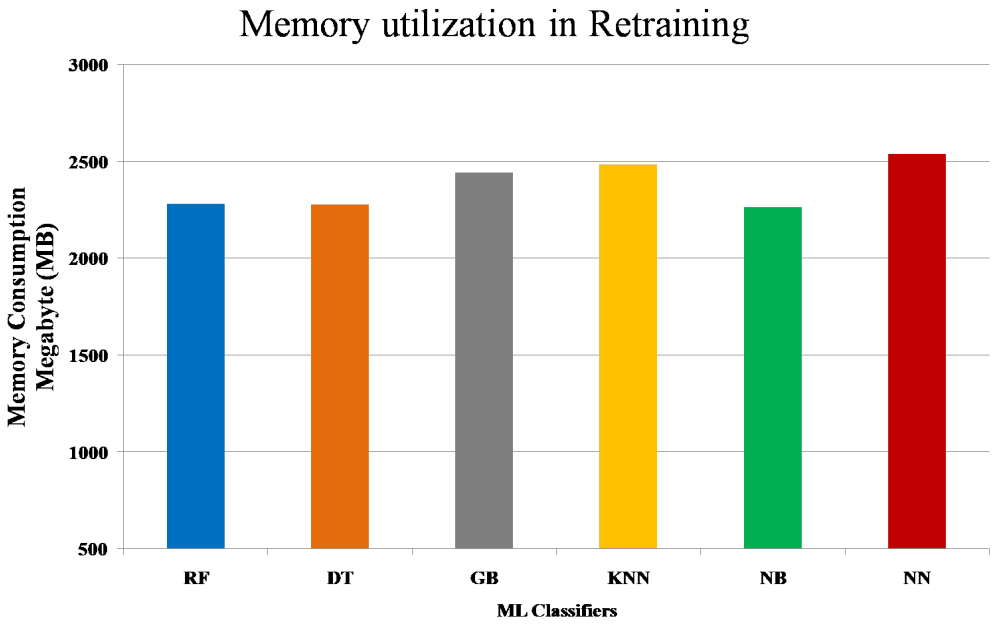


Figure 6.8: Memory consumption during retraining

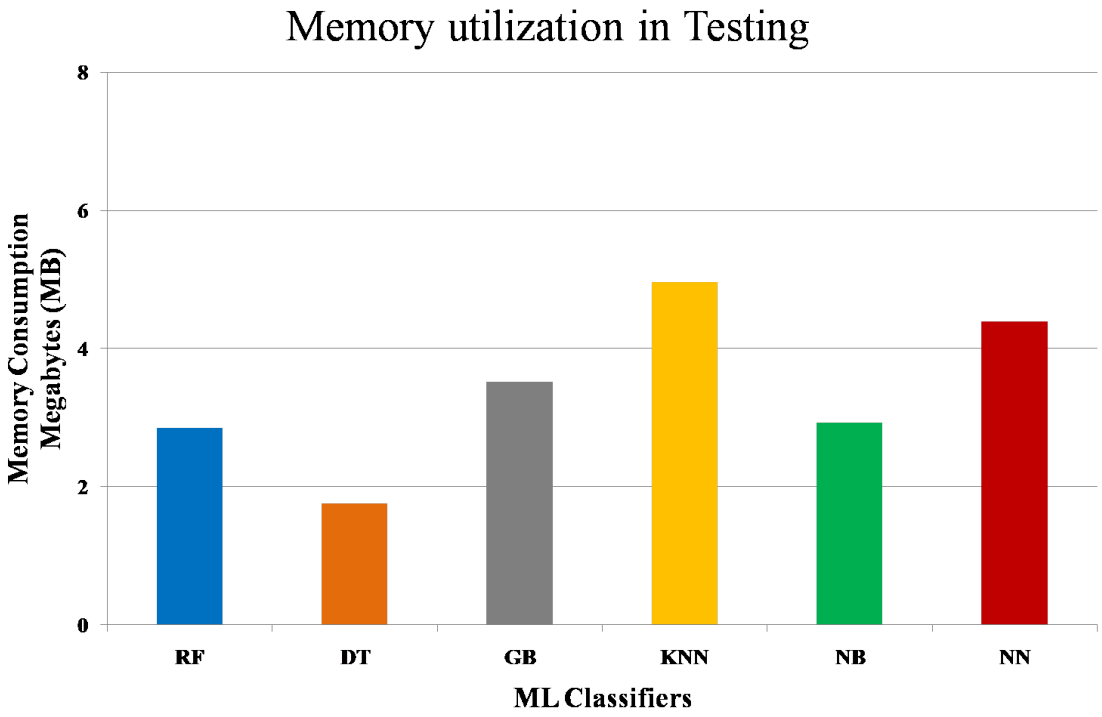


Figure 6.9: Per sample memory consumption after self-healing

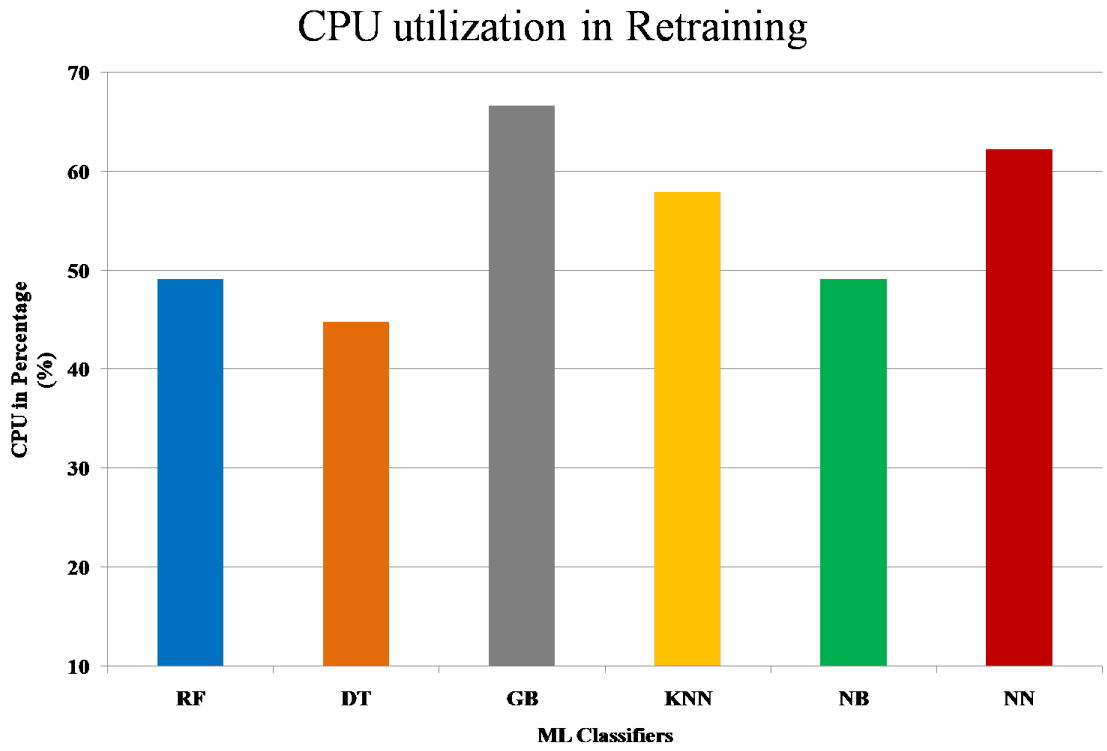


Figure 6.10: CPU utilization during retraining

The usage of the CPU in various models of machine learning techniques in the retraining phase is shown in Figure 6.10. The CPU usage in the DT model is found to be lowest, at an average usage of 45%, which indicates that it consumes less CPU compared to other models. The usage of the RF and NB models in terms of the CPU is moderately higher at about 49%, which is also less. But the usage of the GB model in terms of the CPU is substantially higher at 66.6%, which indicates that the training done in this model requires more CPU. The NN model requires the second-highest usage of the CPU at about 62%, which indicates that the training done in this model also requires more CPU.

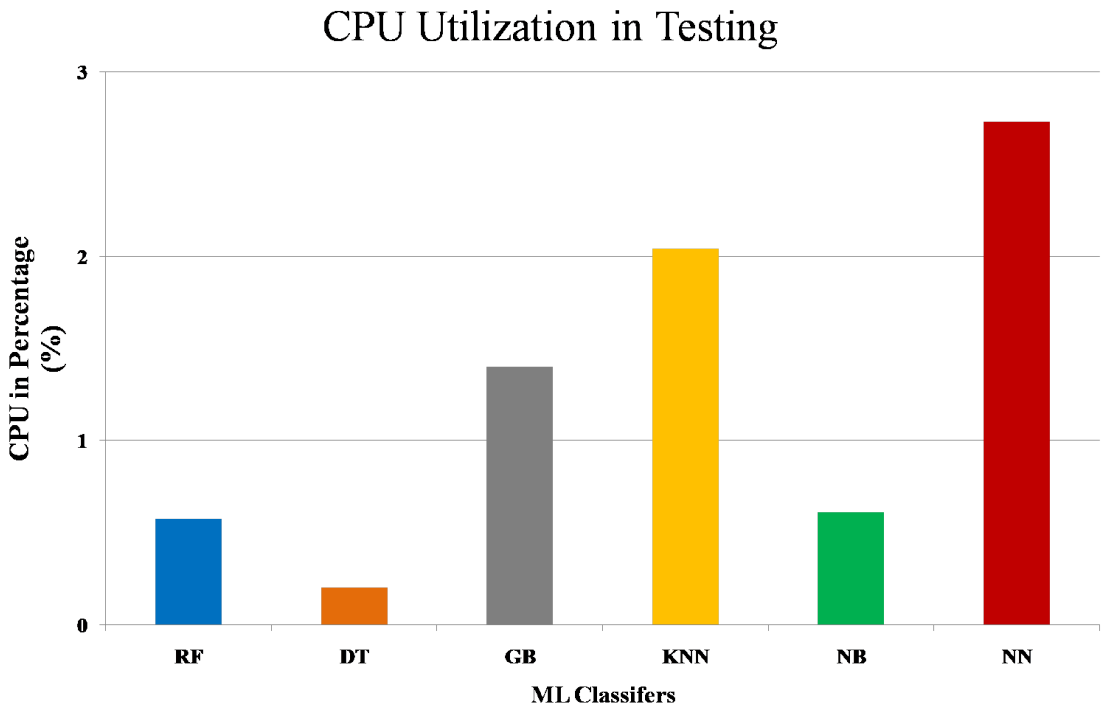


Figure 6.11: Per sample CPU utilization after self-healing

Figure 6.11 showcases CPU usage on different machine learning models during testing. The CPU usage on the DT model is least at 0.2% during testing on a single sample, while it is 0.58% on the RF model during testing on BoT-IoT samples. Low CPU usage on the decision tree and random forest models highlights that these tree-based models consume relatively less CPU as opposed to other models. On the contrary, the kNN model and the NN model consume maximum CPU usage of 2.73%, which is significantly higher than that of other models.

Figures 6.4 to 6.11 offer a clear insight into accuracy, confusion matrix measures, as well as resource consumption parameters, which include time consumption, memory consumption, and CPU consumption. It is clear that the RF, DT, GB, and KNN classifiers are highly efficient in terms of accuracy parameters as they are capa-

ble of distinguishing benign and attack packets effectively. On the other hand, with respect to resource consumption parameters, DT, RF, and NB outperform models like KNN, GB, and NN as far as retraining parameters as well as testing parameters are concerned. Although moderate resource consumption is observed for KNN models while retraining them, their resource consumption escalates to significant values while performing test tasks, thus implying that these models are amongst the most resource-intensive ones.

6.2.6 Evaluating Model Retention using TON_IoT Dataset

In this case, the retrained ML models undergo a comprehensive assessment in terms of testing whether they retain their antecedent knowledge and learn from subsequent exposure to self-learning after undergoing self-healing. This is necessary since failing models may present satisfactory levels of accuracy against new data set examples but still act erratically against TON_IoT examples, which may result in a dramatic degradation of general performance capabilities. As a result, detrimental scenarios may emerge where a self-healed model learns normal attack patterns but still triggers warnings for TON_IoT examples that it accurately detected before self-healing. To objectively establish antecedent knowledge retention in retrained ML models, a comprehensive assessment is undertaken using TON_IoT data set examples. In this case, key performance indicators singled out for assessment entail accuracy, confusion matrix values, testing time, memory usage, and CPU utilization. These indicators offer a holistic viewpoint of performance capabilities of models in question and facilitate a comprehensive understanding of capabilities of models in question

to learn general patterns of data set examples encountered before self-healing.

The performance of machine learning classifiers after the self-healing process for machine learning algorithms is shown in Fig. 6.12, indicating that Random Forest, Gradient Boosting, as well as Neural Network algorithms are capable of maintaining their prior knowledge. The accuracy of RF and GB is impressive at 99.9%, whereas NN is slightly less at 98.56%. RF and GB work by building a collection of a number of independent decision trees, with every decision tree in this collection being trained on a random subset of features and instances from a dataset. One unique parameter in RF and GB is warm_start, which can be used in incremental learning of a model and defined as True when initializing a model.

In this case, warm_start in RF and GB will preserve existing trees in a scenario involving additional instances in a dataset, and thus new trees can be added without affecting or losing existing trees. The unique property or added advantage of this parameter is improved generalization and avoidance of overfitting, especially in a dataset with a large number of dimensions or in noisy datasets. Individual decision trees in this collection or ensemble have learned different things from a dataset and thus make individual predictions for the target variable (0 or 1 in a binary classification problem). For a given input in a trained RF or GB classification algorithm, individual trees in this collection will make individual predictions of a class, and through a majority voting algorithm, a final output or result is determined based on a classification made by a majority of decision trees in this collection.

Like RF and GB, NN is also capable of doing incremental learning by adjusting

itself according to new datasets without undergoing retraining. The NN starts with an existing model to avoid loss of existing knowledge while modifying some layers with a lower learning rate.

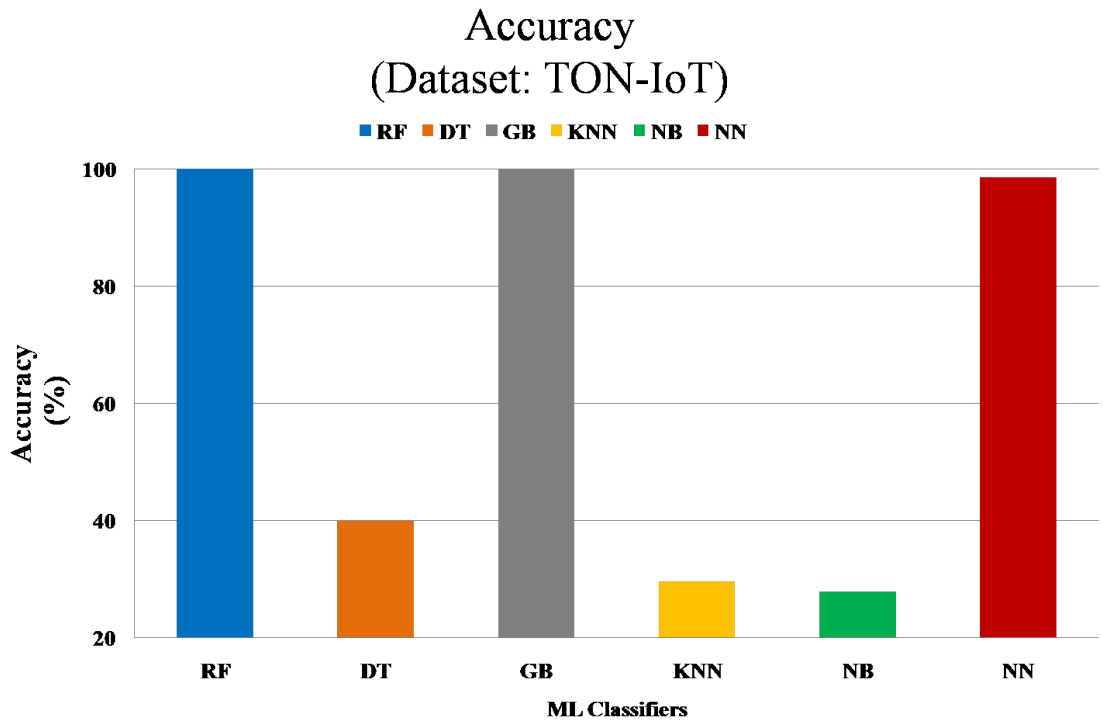


Figure 6.12: Evaluating model retention and accuracy on the TON_IoT Dataset

Conversely, the performance of classifiers like DT, KNN, and NB degrades after the self-healing phase. This indicates that the learned knowledge from the TON_IoT dataset is not retained well during the retraining phase using the BoT-IoT dataset. This is because none of these classifiers inherently possesses the ability for light learning [55]. This means that when more data are added, the decision tree needs to be rebuilt altogether because the decision tree model is static. For the NB classifier, it recalculates all the probabilistic values based on the entire dataset. This makes it inefficient for light learning. For KNN, it is a type of lazy learner

that stores all the training samples. However, the addition of more data results in the prediction favoring the latter or more prominent samples, thus downgrading the impact of the former TON-IoT dataset.

In Figure 6.13, the results of the machine learning models on the confusion metric parameters: TPR, TNR, FPR, and FNR. Talking about TPR first. RF and NN models with high TPR are very efficient in identifying positive instances correctly. GB also performs well in this case. But models such as DT, KNN, and NB have comparatively less TPR value. Hence more chances of missing positive instances. Talking about TNR. RF and NN models give very high results. Hence very efficient in identifying negative instances correctly. GB results in moderate TNR. But models such as DT, KNN, and NB result in low TNR. Hence more chances of incorrect classification of negative instances. Now moving towards FNR. Since FNR results in better accuracy. Hence models such as DT, KNN, and NB result in higher values, whereas models such as RF, GB and NN result in less FNR. Hence very efficient in identifying positive instances. In case of FPR models such as RF, GB, and NN result in less FPR whereas models such as DT, KNN, and NB result in comparatively more FPR.

Figure 6.14 shows the time required for testing for samples of various ML classifiers. The figure indicates the presence of large variations in computational efficiency. The minimum time required for testing is for the DT classifier, taking only 0.26 ms. The second most efficient is the NB classifier with testing time of 0.33 ms. Similarly for RF classifier, the time required is only 0.047 ms. On the other hand, the maximum testing time is for the KNN classifier, which is 334.39 ms.

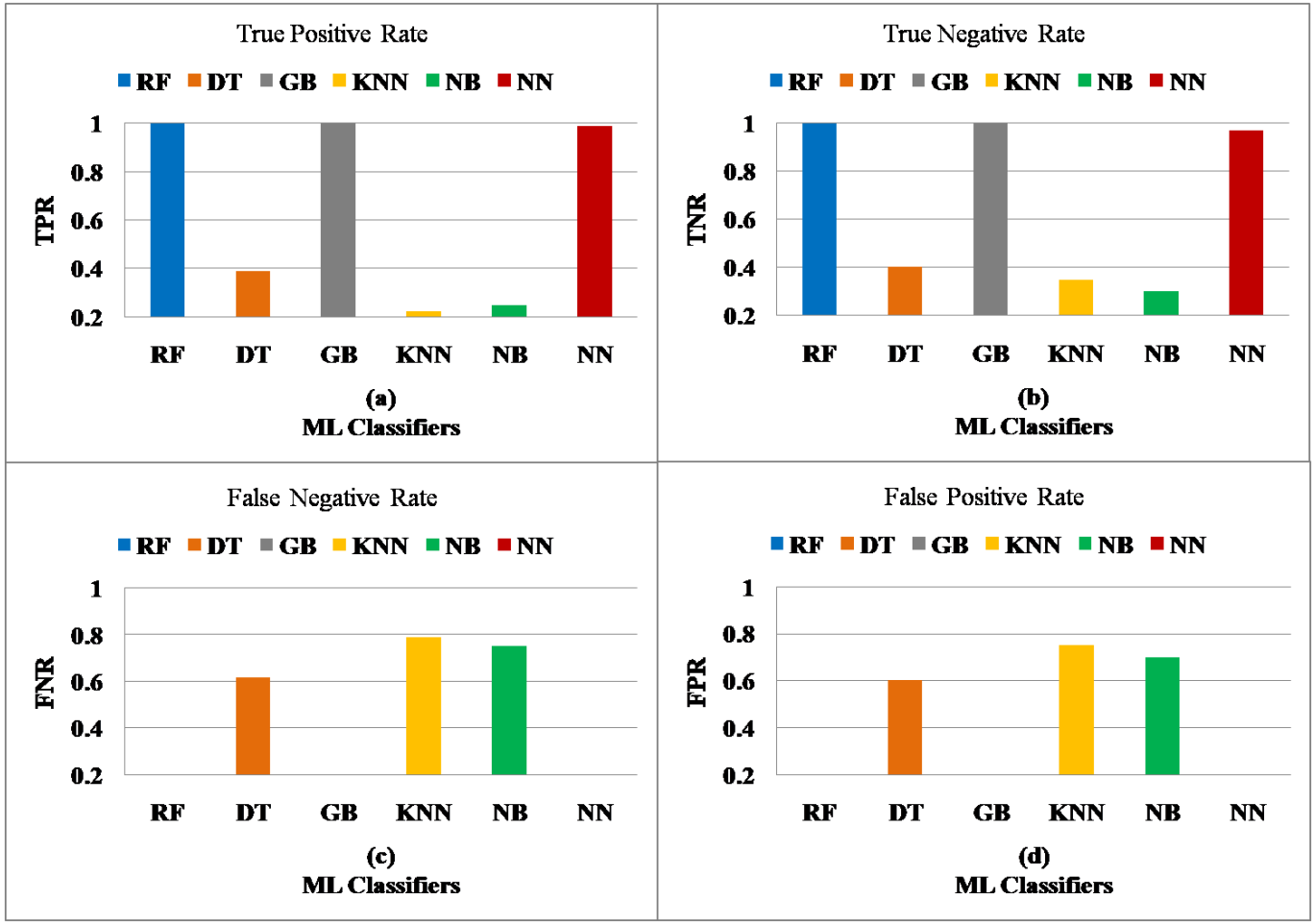


Figure 6.13: Performance metrics (a) TPR (b) TNR (c) FNR and (d) FPR: TON_IoT dataset

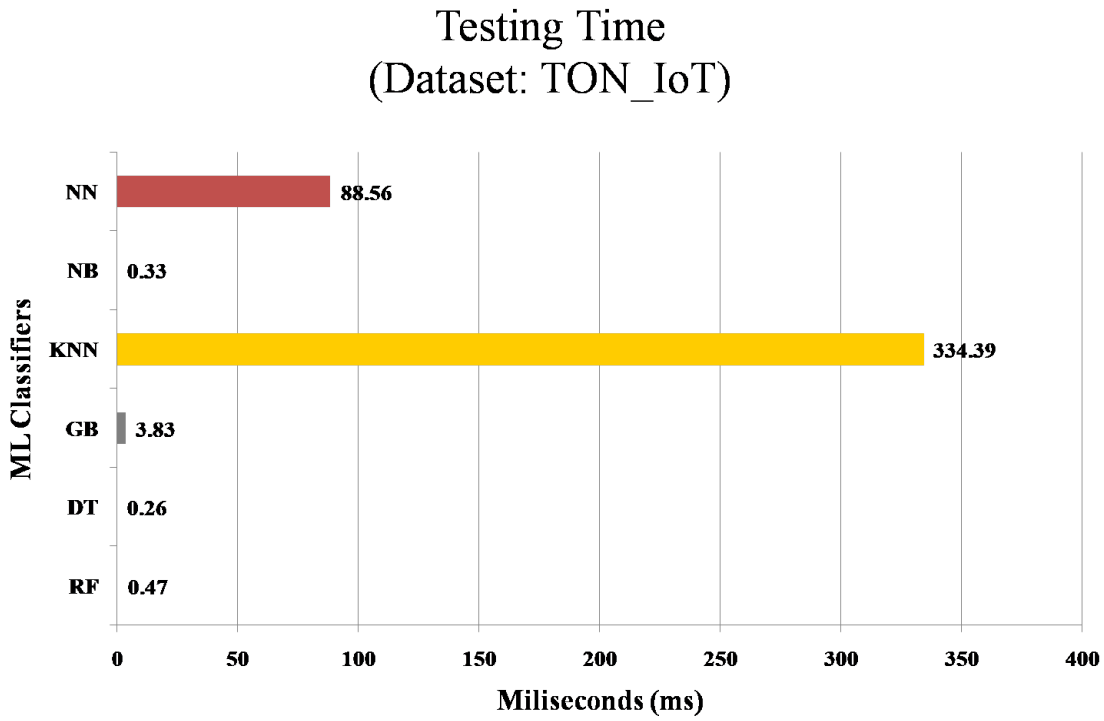


Figure 6.14: Per sample testing time after self-healing using TON_IoT dataset

The second highest time is 334 ms. It is understandable since the classifier computes distance for all samples. NN is the second least efficient classifier. It records 88.56 ms. The GB classifier has moderate performance with testing time of 92 ms. The testing times shown in Figure 6.14 for TON_IoT and Figure 6.7 for BoT-IoT are relatively equal. The two datasets have samples of network traffic, and when the IDS model is tested using the same features, preprocessing techniques, and classifiers for both datasets, the testing times will likely be equal [24]. This shows that when the testing times count towards the performance of the IDS-model, there will be little difference between the two datasets.

Figure 6.15 below shows the CPU and memory consumption of various ML classifiers during testing. The most efficient use of resources is found in NB, which

consumes only 0.15% CPU and 1.23 MB of memory, making it the least resource-hungry of all tested classifiers. This is followed by DT, which also consumes 0.2% CPU but 1.75 MB of memory for each record. Next comes RR, which consumes 0.71% CPU with 3.3 MB of memory, though not as low in efficiency as the preceding two but marginally higher in consumption of both CPU and memory. However, NN and GB consume significantly higher levels of both CPU and memory during testing. The memory consumption of NN is highest at 7.68 MB, with both NN and GB taking the heaviest toll in CPU at 3.4% (NN) and 4.7% (GB), attributed to multi-layered and iterative algorithms of NN and GB, respectively, which are memory-intensive and CPU-hungry, respectively.

However, it is important to state that the test time of the RF, NN, and GB classifiers increased significantly because of the complexity involved in these classifiers. The complexity involved here comes from the fact that these classifiers were trained again on the BoT-IoT dataset with the inclusion of the TON_IoT dataset. On the other hand, the KNN, NN, and NB classifiers demonstrated a constant usage of system resources because they simply overwritten the knowledge gained from the TON_IoT dataset with knowledge gained from the BoT-IoT dataset.

The results showing the outcomes of the evaluation of the re-trained models of ML on the samples from the TON_IoT dataset are shown in Figures 6.12-6.15 below. As seen, RF, GB, and NN perform better with respect to accuracy, TPR, FNR, TNR, and FPR, while DT, KNN, and NB perform poorer with respect to accuracy with higher values of FNR and FPR.

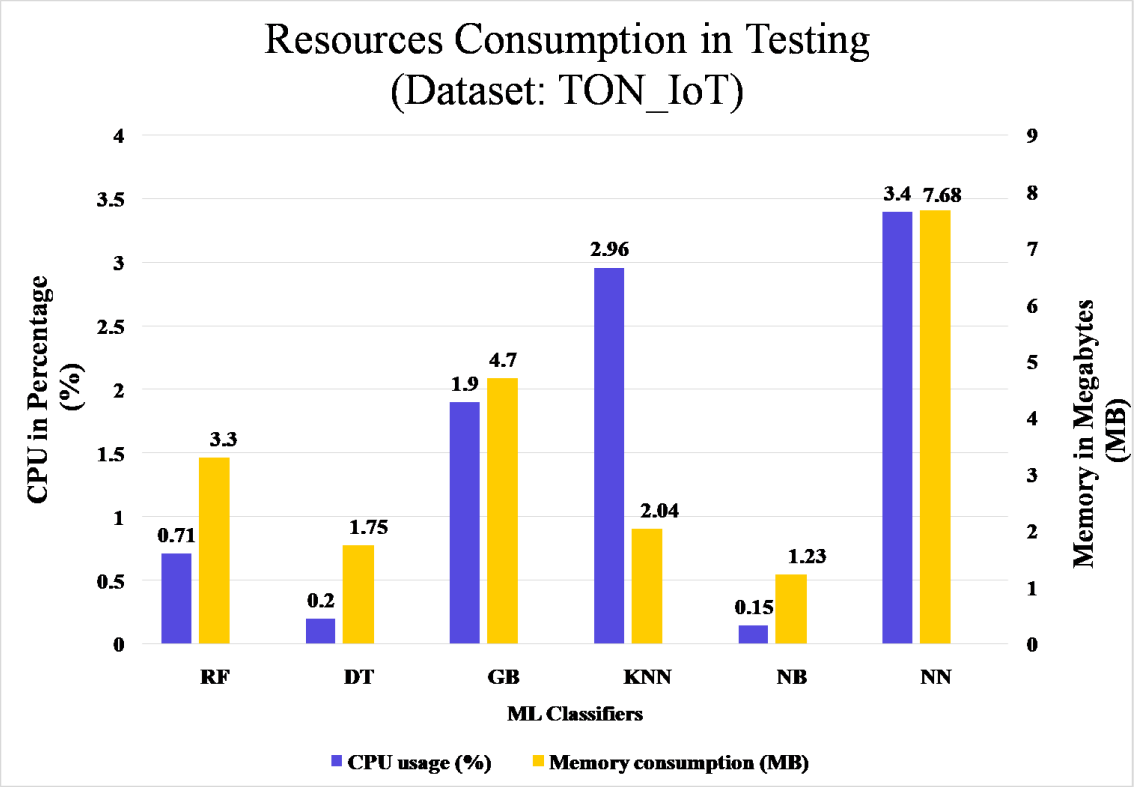


Figure 6.15: Per sample resources utilization after self-healing using TON_IoT dataset

In the area of resource usage, GB and NN are observed to be relatively less efficient because of their complex structures that require high CPU and memory usage; hence, real-time application efficiency is limited. On the contrary, RF has an advantageous resource profile in terms of low CPU and memory usage and very low testing time; hence, efficient decision-making in a real-world network setting is observed. Although DT and NB also have relatively low resource usage and fast testing time, their poor performance in accuracy and classification measures (FNR, FPR, TPR, and TNR) reduces their relative efficiency compared to RF.

6.3 Summary

This chapter presents an overall analysis of the proposed SHIoT outlined in Chapter 5. The findings indicate that self-healing through retraining is an effective mechanism to maintain the capacity of classifiers in distinguishing and assigning newly encountered as well as previously learned data sources into predefined categories. The performance analysis is carried out using the BoT-IoT and TON_IoT datasets in order to classify the performance of six machine learning classifiers. After retraining and testing the models using the BoT-IoT dataset, the accuracy levels indicate that RF, NB, GB, KNN, and DT perform better than others with 99.9% accuracy rate. The performance of the retrained models is then analyzed using the TON_IoT dataset to check if any knowledge is retained or overwritten.

The above results reveal that RF, GB, and NN maintain their learning, while DT, NB, and KNN fail to do so. Moreover, the above analysis shows that RF, NB, and DT perform the most efficient usage of resources. However, both NB and DT perform poorly with 27.8% and 39.9% accuracy, respectively. On the contrary, RF has an accuracy of 99.9% with 0.47 milliseconds of testing time per sample, 3.3 MB of memory, and 0.71% of CPU resources per sample for the TON_IoT dataset. Although GB and NN have achieved 99.9% and 98.5% accuracy, respectively, their increased resource utilization in terms of testing time, CPU, and memory makes them less preferable for usage in IoT devices. Hence, RF outperforms all classifiers for both datasets and is ranked as the best performing classifier.

Talking about accuracy with comparison to the method introduced in [64],

our proposed method SHIoT provides a significantly higher accuracy of 99.9%, while for [64], there is an additional cost of computation as they have used RNN. In a study [153], they have provided an accuracy of only 90% with infected devices being isolated after 15 seconds with a detection time of 3.1 seconds. SHIoT not only outperforms them but also is more efficient with an analysis time of only 0.47 ms to predict a result for a sample.

The self-healing process further augments the efficiency of the IDS by ensuring better adaptability to novel data. Although SHIoT achieves fast prediction speed, the self-healing process may be computationally expensive. This could be a potential limitation in its application in an IoT setup. In this regard, the next chapter explores the incorporation of ELIDS with SHIoT. The resultant approach aims to ensure better adaptability to novel data with an efficient utilization of resources; hence, an efficient IoT security system would be developed.

CHAPTER 7

Integration of Ensemble Feature Selection with Self-healing Mechanism

7.1 Introduction

This chapter discusses the analysis of the integration of the Ensemble feature selection for Lightweight Intrusion Detection Systems with a Self-Healing Mechanism (SHIoT-ELIDS). The proposed scheme integrates two major components: a self-healing mechanism and a lightweight ensemble feature selection scheme for efficiently optimizing the detection efficiency with low computational complexity. The feature selection scheme is discussed in detail in Chapter 3. In addition, Chapter 5 discusses in detail the mechanism associated with the self-healing approach to ensure that the IDS system can heal itself automatically in the event of any interruption or attack.

After incorporating these two mechanisms into the SHIoT-ELIDS framework,

an exhaustive tests were carried out in order to validate its efficiency and robustness. The test initiated involved the usage of ten individual dataset files from the TON_IoT repository, which included information relevant to DoS/DDoS attacks. These datasets encompassed various instances to ensure an in-depth evaluation of the IDS efficiency in different scenarios. This detailed test will ensure that the IDS features are apt for handling complex security needs in an evolving IoT environment.

7.2 Proposed Integrated Framework for Lightweight IDS with Self-healing Capability

The design of the lightweight IDS with self-healing functionality is shown in Figure 7.1. As shown in Figure 7.1, the design involves two separate flows, which are the normal flow and the self-healing flow. Every IoT device has a health-monitoring module that monitors the resources and provides a danger signal accordingly. The IoT gateway in the design includes vital components such as the IDS with machine learning capabilities, the alert-receiver module, and the packet capture storage, which facilitates smooth communication and allows for the self-healing flow. Those processes that consume a lot of resources such as packet clustering, selection of features for the machine learning models for training purposes, data pre-processing, and re-training of the models are done in the Fog devices, which have better calculation capabilities than the IoT devices to avoid straining them.

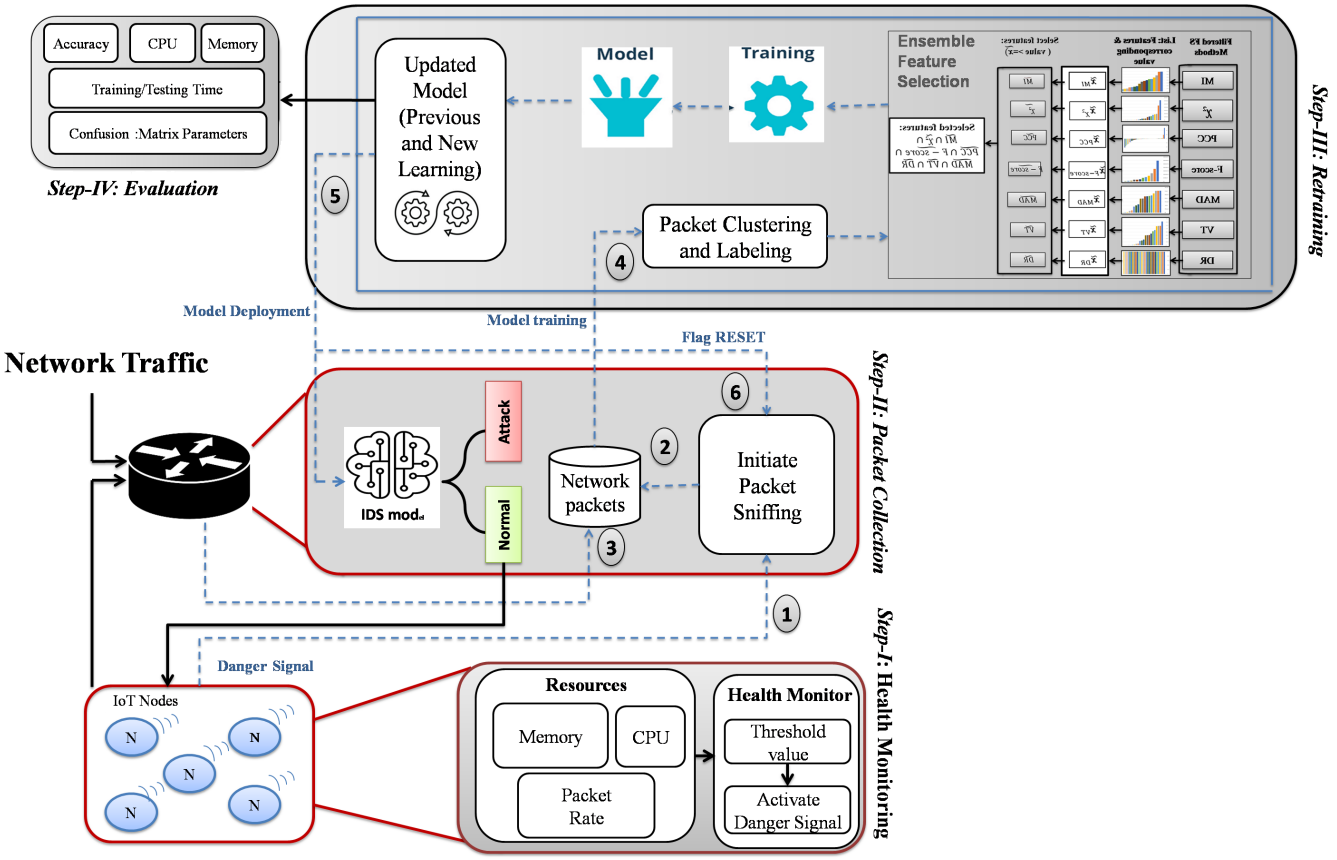


Figure 7.1: Proposed framework for lightweight and self-healing IDS

7.2.1 Normal Flow

The normal flow of operations is shown in Fig. 7.2. This starts with an offline training phase where the TON_IoT dataset undergoes processing using an ensemble FS approach presented in Chapter 3. This is followed by using the selected features for training an RF classifier. This choice is guided by results presented in Chapter 6, highlighting its superiority over other classifiers like DT, KNN, NB, GB, and NN, not only in-domain testing, but also for its performance in cross-domain testing. Moreover, its ability to incrementally add new trees to its forest during retraining while retaining its previously acquired knowledge is also an added advantage, presented in Chapter 6. Its lower computational requirement compared to GB and NN also makes it an attractive solution for resource-scarce environments like IoT.

After the training phase is completed, the developed model is placed in the IoT gateway. In the IoT gateway, the developed model analyzes the network traffic in real time and helps classify the traffic as either malicious or benign. The benign traffic is passed through the IoT network without any issues, while the anomalous traffic is discarded.

7.2.2 Flow Directions in Self-Healing Mechanism

The self-healing flow shown in Figure 7.3 responds to the attack patterns seen in practice, which are called emergent attacks. If the normal flow pattern fails to identify the malicious packet, which happens when attackers apply evasion and encryption techniques, this packet could gain entry into the Internet of Things network

with a possible effect of consuming resources and damaging Internet of Things nodes.

For this purpose, the IoT nodes produce a danger indicator in case of depletion of resources. The indicator is sent to the packet sniffing module, which in turn instructs the tcpdump utility to start the sniffing process and also sets a flag to 1, which denotes the beginning of the self-healing process.

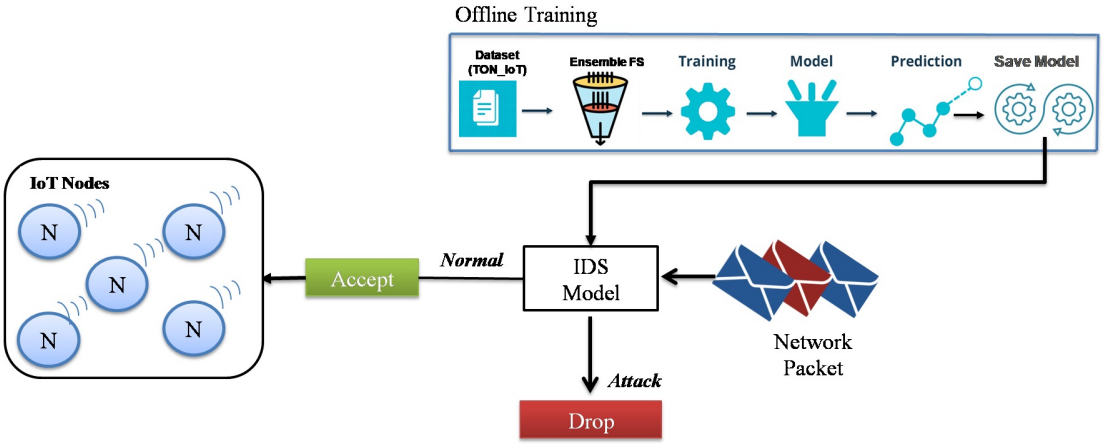


Figure 7.2: Normal flow

Additionally, the captured packets are sent to the clustering and labeling stage. The labeled dataset produced as a result of this process is then used for the retraining of the model, thereby allowing the IDS to learn and grow with time as it encounters new threats. After completing the process of retraining, the resulting model is then implemented in the IoT gateway, following which a signal is sent to the packet-sniffing module to set the flag back to 0, thus denoting the completion of the self-healing process in the context of the current attack incident and readiness of the packet-sniffing module to receive future danger signals.

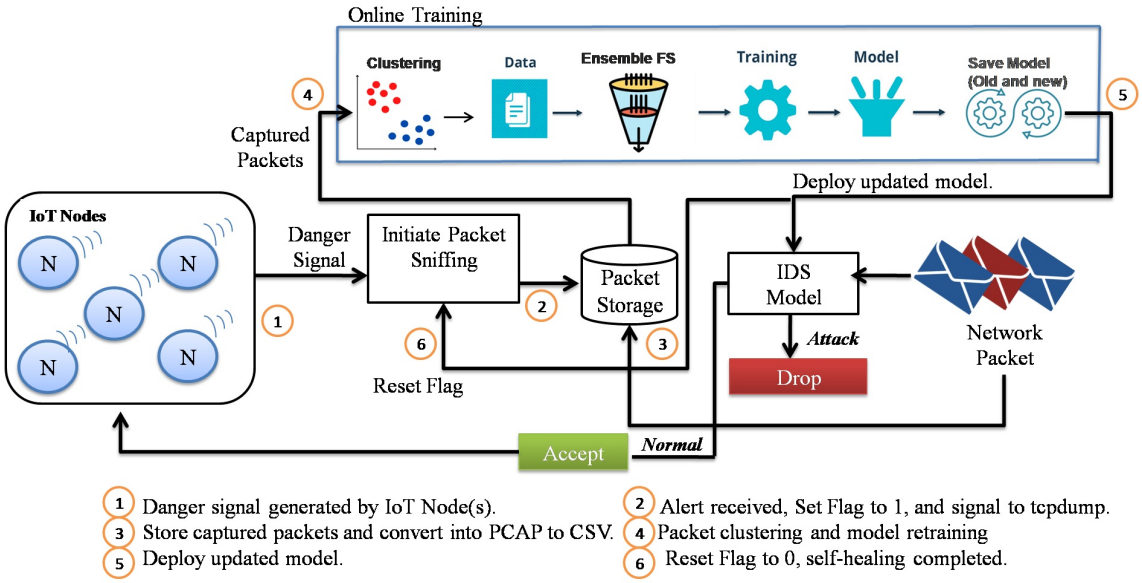


Figure 7.3: Flow direction in self-healing

7.3 Performance Evaluation of Proposed SHIoT-ELIDS

This section evaluates the performance of the proposed lightweight with a self-healing IDS. In this Chapter, all experiments are performed using the same tools and techniques used in Chapter 4.

7.3.1 Performance Evaluation of Trained RF model

The initial training phase is performed utilizing the Train_Test_Network subset from the TON_IoT dataset, as discussed in Chapter 4. The produced model is shown in Figure (see Appendix A). This graph clearly reveals that the trained model uses a combination of decision trees, implementing binary classification tasks utilizing the following features: 'ts', 'src_port', 'dst_ip', 'dst_port', 'protocol',

'conn_state', and 'service', among others.

As shown in Figure (Appendix A), the level-0 tree starts by fixing the root node's class as conn_state, where the split criterion is conn_state 0.056. But since the Gini impurity equals 0.5, it implies a high level of ambiguity, knowing that the 40,487 samples passed to this node are evenly split between the two classes with 24,214 in class 0 and 16,366 in class 1. Moreover, it predicts class 0, which is a majority class, as the root node, starting the process of splitting in the classification task. For level 1, the split on the first node (the left child of the root node) is on feature dst_ip, with the splitting criterion set as dst_ip 0.002. However, the Gini impurity reduces somewhat to 0.08, reflecting an improvement in splitting classes, where 16,273 samples belong to class 1, with only 93 samples belonging to class 0.

The trained random forest classifier is tested on a variety of unseen dataset files such as Network_1, Network_9, Network_10, Network_11, Network_13, Network_14, Network_15, Network_16, Network_17, and Network_18 collected from the publicly available TON_IoT dataset archive in both CSV and PCAP file formats {[188]}. The datasets include records of both benign traffic and DoS/DDoS attacks, summarized in Table 7.1. All instances in these datasets are treated as unseen by the IDS system since they are not used in training the IDS system. Testing the IDS system using the datasets would give a complete analysis of the ability of the system in dealing with novel attacks.

Table 7.1: Instance distribution across datasets

Dataset	Instances		Total Instances	Percentage of Dataset (%)
	Normal	Attack		
Network_1	208,679	0	208,679	2.4
Network_9	24,739	975,261	1,000,000	11.5
Network_10	30,002	969,998	1,000,000	11.5
Network_11	35,168	839,637	874,805	10.1
Network_13	917	999,083	1,000,000	11.5
Network_14	583	999,417	1,000,000	11.5
Network_15	1,060	998,940	1,000,000	11.5
Network_16	1,891	998,109	1,000,000	11.5
Network_17	33,711	966,289	1,000,000	11.5
Network_18	49,436	563,440	612,876	7.1

The results for accuracy shown in Figure 7.4 indicate that the IDS model, with the Train_Test_Network file, performs well on some data files. Indeed, the system achieves an accuracy level of 96.5% on the Network_1 file and 81% on the Network_13 file. However, its performance is poor on the other datasets, with an accuracy level on the Network_9 file being significantly low at 46%. The poor accuracy levels on the other files indicate that the system lacks functionality in being able to generalize the networks with varied traffic, with zero-day attacks being common on such networks and thus difficult for the system to accommodate.

Further, as seen in Figure 7.5, the values for specificity are 1.0 for Train_Test_Network files. For files with Network_1, Network_13, Network_17, and Network_18 files, specificity reduces by less than 0.5. Here, the system is still able to pick out most of the benign items but is not able to achieve perfect specificity.

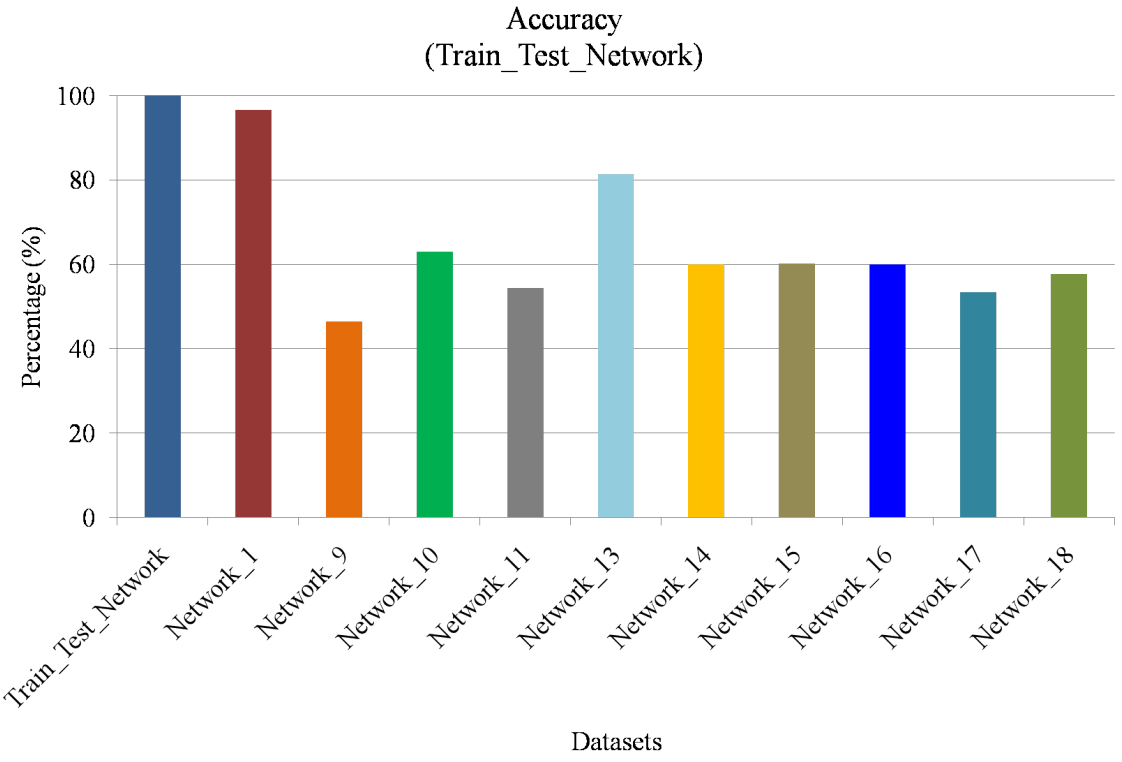


Figure 7.4: RF accuracy using Train_Test_Network Dataset

Regarding the recall metric, it is noted that the recall for the Network_1 file is 1, ensuring it holds only benign samples. On the other hand, a recall of 1 is achieved by the Train_Test_Network dataset, signifying ideal attack pattern detection. For Network_9, recall achieved is 0.42, the lowest among the analyzed files. The noted degradation in the performance of the Random Forest (RF) method on various data files implies a reduced ability to detect all attack samples, making IoT devices more vulnerable to DoS/DDoS attacks. Additionally, it is noted that the RF method on the Train_Test_Network dataset takes only 0.0021 ms to process an example. Regarding resources, it is stated that the method consumes 0.0722 MB memory and 0.0006% CPU per processed example. Note that, since the method is an offline solution, the time taken to train is irrelevant for performance analysis in terms of

real-time processing. Also, it is noted that the trained method consumes only 426 MB, ensuring smooth deployment on IoT gateways with restricted resources.

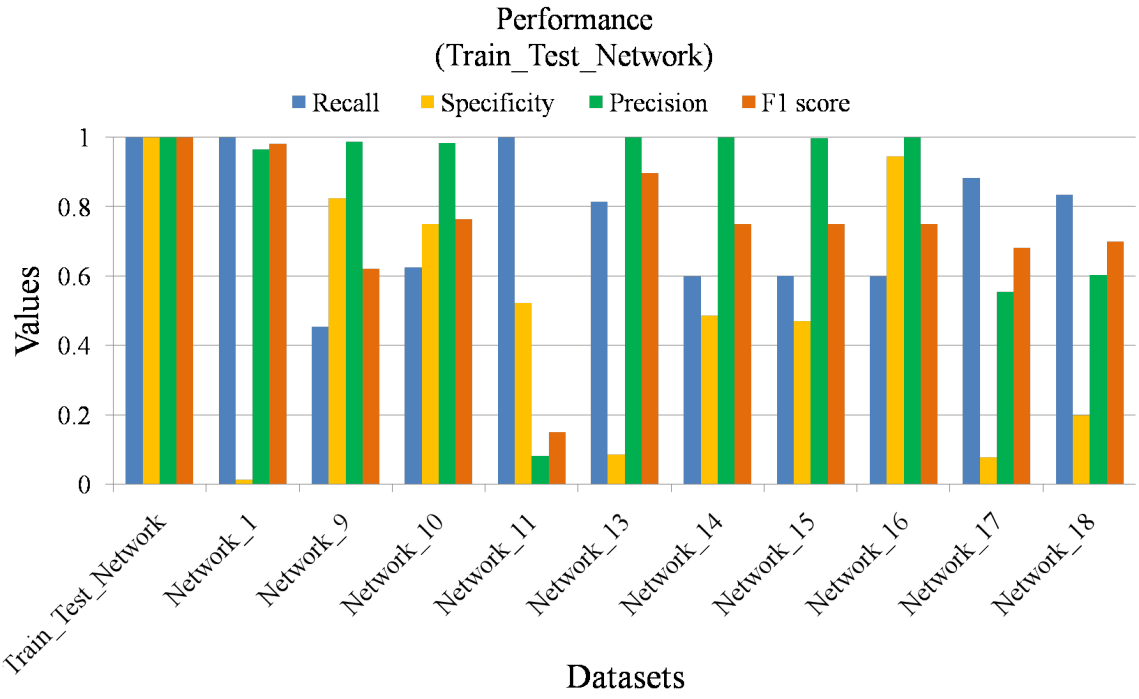


Figure 7.5: RF performance using Train_Test_Network Dataset

7.3.2 Model Retraining using Network_9 Dataset

On the basis of the results demonstrated in Section 7.3.1, it is clear that the Network_9 dataset has the poorest accuracy. On the premise of the results achieved by the self-healing process, the data in the Network_9 file is utilized. The PCAP file associated with Network_9 is then converted to CSV format and then further processed by the K-means clustering module. The resulting clusters of this module are illustrated in Figure 7.6, with the varying sizes of the clusters specifying that there is a greater amount of attacks in the file compared to benign examples.

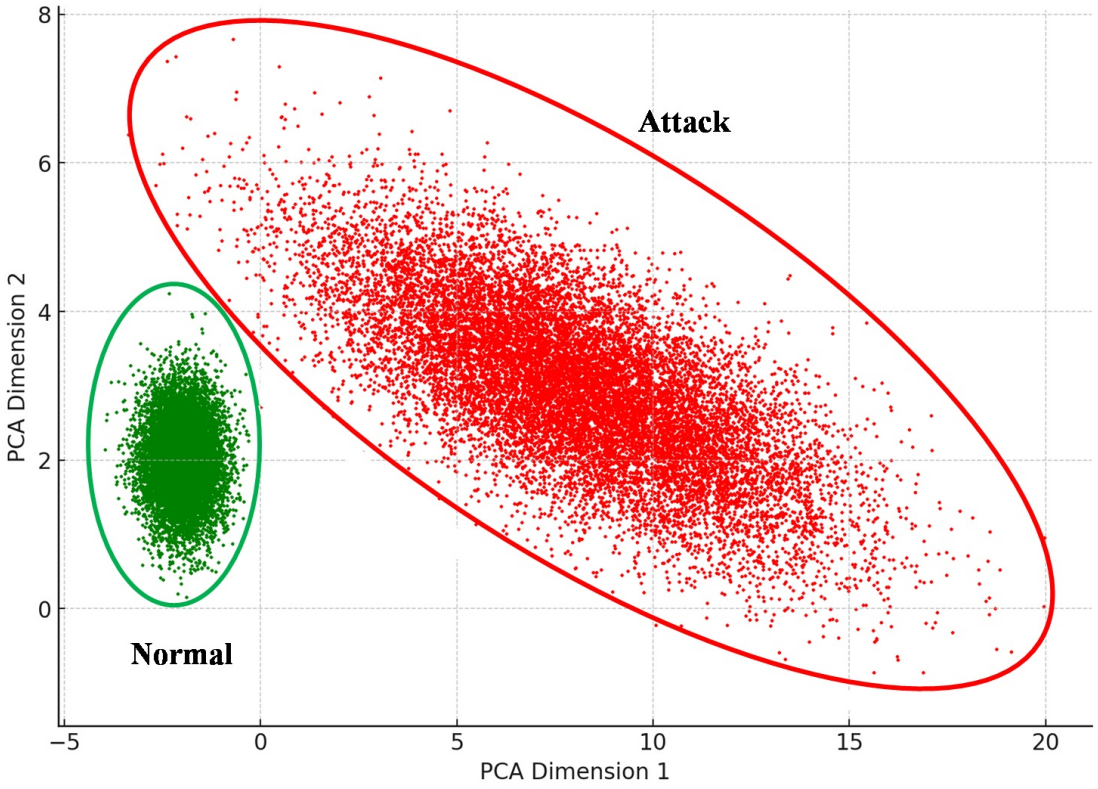


Figure 7.6: K-means clustering for Network_9 Dataset

After clustering and labeling, 74,208 samples are chosen to be input to the ensemble FS algorithm, which is used to determine the set of most relevant features from the pool of 43 features, as outlined in Table 7.2. Table 7.2 lists the ranking of the features by the ranking score assigned by the ensemble FS algorithm using the output from seven different FS algorithms. Features with importance scores above 22 (the median importance score) are retained; details are provided in Table 7.3. The rest are eliminated. The ensemble FS approach finalizes the top eight features to be used to retrain the RF classifier on the Network_9 dataset; these features are identified in Table 7.3. These features include all those that were used to train the RF on the Train_Test_Network dataset with dst_pkts incorporated. The eight features obtained by the ensemble FS method are then used to retrain the RF

algorithm on the Network_9 dataset. The accuracy of the retrained RF algorithm is then tested on the Network_9 set alone and on further unseen datasets.

The RF model is shown in Fig. (in Appendix B), which takes the Network_9 dataset file as input and demonstrates decision-making according to important variables like ts, src_port, dst_ip, dst_port, protocol, conn_state, dst_pkts, and services. The conn_state attribute is a critical piece of information in the root node of the tree and shows a Gini impurity of 0.035 since it represents a large portion of a clean split of classes. Other subsequent nodes split classes even more according to criteria like service and dst_ip; splits eventually tend toward a Gini index of 0 since they represent completely sure regions of the split. The leaf nodes in this case represent final classifications of observations that depend on classes of a majority of a split portion of a set. (For a better understanding, see Fig. in Appendix B.) Based on the results of accuracy in Figure 7.5, there is an indication that the retrained intrusion detection system IDS model performs better than the pretrained model with an accuracy rate of well over 96% in all files of the dataset. This shows that the model is capable of leveraging the knowledge that it got during the training phase in a way that helps the model perform better in accurately predicting both the benign as well as the malicious examples in the dataset.

Table 7.2: Ranking and corresponding values of Network_9 Dataset features set

Feature Rank	IG	Value	chi2	Value	F-score	Value	R	Value	VT	Value	MAD	Value	DR	Value
1	src_ip	0.84	dns_RD	7433.537352	ts	35687.51028	dst_port	0.62472	proto	0.17244	ts	0.36732	ts	1.2011184
2	ts	0.83	ts	5438.69527	dst_port	21552.1968	dns_rcode	0.2274	src_port	0.14112	src_port	0.36612	weird_notice	1.2007752
3	src_bytes	0.68	service	2299.514294	proto	14053.8498	duration	0.03084	dns_RD	0.14052	conn_state	0.36516	conn_state	1.2002412
4	src_ip_bytes	0.59	src_port	2147.52713	src_port	9701.03484	conn_state	0.02904	conn_state	0.1386	dns_RD	0.3228	dst_port	1.2001968
5	conn_state	0.53	dns_rcode	1961.735359	dns_AA	7442.97048	ts	0.01188	dns_query	0.10992	service	0.28872	ssl_resumed	1.2001872
6	dst_port	0.52	dns_qtype	1481.85601	service	6156.28944	proto	0.0114	dns_rejected	0.10164	proto	0.279	http_version	1.200156
7	duration	0.48	dns_AA	1342.273049	dns_rcode	3574.7166	src_ip	0.0114	service	0.0846	dst_pkts	0.2742	src_bytes	1.2001272
8	dst_ip	0.44	proto	1118.175219	dst_ip	2923.46592	service	0.00492	ts	0.07368	dns_rejected	0.2718	ssl_version	1.200126
9	src_port	0.41	dst_ip	594.5205933	dns_qtype	2870.25624	dst_ip	0.0042	dns_RA	0.07284	dns_RA	0.23568	weird_name	1.20012
10	dst_pkts	0.32	dns_query	533.77936	dns_RA	1104.52248	src_port	0.00168	dns_AA	0.07176	dns_AA	0.23424	dst_ip	1.2000804
11	src_pkts	0.22	dns_RA	521.5680153	dns_query	859.72968	src_bytes	0	dst_port	0.04116	dst_port	0.1392	http_resp_mime	1.2000732
12	dst_bytes	0.20	dns_qclass	292.4811048	conn_state	489.17256	http_resp_mime	-0.00084	dst_ip	0.03384	dst_ip	0.12144	ssl_cipher	1.200066
13	dns_query	0.18	dst_pkts	188.6556213	dns_qclass	294.92472	http_user_agent	-0.00252	dst_pkts	0.03084	dns_rcode	0.10044	http_user_agent	1.2000648
14	dst_ip_bytes	0.17	conn_state	125.2734444	weird_notice	189.42336	src_pkts	-0.0042	dns_qtype	0.01116	dns_qtype	0.03684	src_port	1.2000648
15	dns_rejected	0.12	weird_addl	80.4	dst_pkts	115.9836	http_resp_body	-0.00432	src_ip	0.01008	src_ip	0.02748	http_method	1.2000408
16	proto	0.11	ssl_resumed	64.88275862	ssl_resumed	106.2462	http_status_code	-0.00444	dns_qclass	0.0036	dns_qclass	0.00732	dst_bytes	1.2000276
17	dns_RD	0.09	ssl_established	62.12268908	ssl_established	98.12796	dst_pkts	-0.0048	weird_notice	0.00252	weird_notice	0.00504	src_ip_bytes	1.2000144
18	dns_qtype	0.08	http_trans_depth	43.68392336	weird_name	84.8274	src_ip_bytes	-0.00588	weird_addl	0.00072	weird_addl	0.00204	dst_pkts	1.2000144
19	service	0.04	weird_name	41.41745888	duration	63.6852	http_req_body	-0.0066	ssl_established	0.0006	ssl_established	0.0018	ssl_issuer	1.2000108
20	dns_rcode	0.03	ssl_version	35.26666667	dst_bytes	56.1006	http_orig_mime	-0.00708	ssl_resumed	0.00048	ssl_resumed	0.00168	proto	1.2000108
21	dns_AA	0.03	dst_port	28.77966978	ssl_version	43.81992	http_uri	-0.00792	http_version	0.00048	weird_name	0.00132	ssl_subject	1.2000096
22	dns_RA	0.02	ssl_cipher	17.87076923	dns_RD	42.82896	http_trans_depth	-0.00804	ssl_version	0.00036	src_bytes	0.00108	service	1.2000096

23	missed_bytes	0.01	dst_bytes	17.50022419	ssl_cipher	29.80608	http_method	-0.00864	src_bytes	0.00036	http_version	0.00096	http_req_body	1.2000096
24	dns_qclass	0.01	http_version	9.035294118	missed_bytes	9.33312	http_version	-0.01164	weird_name	0.00036	ssl_version	0.00096	missed_bytes	1.2000084
25	ssl_issuer	0.00	dns_rejected	5.04098193	http_version	9.03972	ssl_cipher	-0.02112	http_status_code	0.00024	http_status_code	0.00072	src_pkts	1.200006
26	weird_name	0.00	missed_bytes	4.238360639	dns_rejected	8.73312	dns_RD	-0.02532	http_resp_mime	0.00024	ssl_cipher	0.0006	duration	1.200006
27	ssl_version	0.00	http_orig_mime	3	src_ip	8.70804	ssl_version	-0.02568	http_user_agent	0.00024	http_uri	0.0006	http_resp_body	1.2000048
28	ssl_established	0.00	http_req_body	2.689855072	http_method	5.02812	weird_name	-0.03564	ssl_cipher	0.00024	http_resp_mime	0.0006	dst_ip_bytes	1.2000048
29	http_version	0.00	http_uri	2.656939203	http_trans_depth	4.36488	ssl_established	-0.0384	http_uri	0.00024	http_user_agent	0.0006	dns_qclass	0
30	http_resp_mime	0.00	http_method	2.568888889	http_uri	4.19064	ssl_resumed	-0.03984	http_method	0.00012	http_method	0.00048	http_uri	0
31	weird_notice	0.00	ssl_subject	1.428571429	http_orig_mime	3.33348	weird_addl	-0.04164	dst_bytes	0.00012	dst_bytes	0.00048	src_ip	0
32	ssl_cipher	0.00	src_bytes	1.25	http_req_body	2.92932	weird_notice	-0.05328	src_ip_bytes	0	duration	0.00024	http_trans_depth	0
33	ssl_subject	0.00	http_resp_body	1.205605144	src_ip_bytes	2.33628	dns_qclass	-0.06636	dns_rcode	0	src_ip_bytes	0.00024	http_orig_mime	0
34	http_method	0.00	ssl_issuer	1.08	ssl_subject	1.57896	dst_bytes	-0.08544	ssl_issuer	0	http_trans_depth	0.00012	ssl_established	0
35	http_uri	0.00	src_ip_bytes	0.995862327	weird_addl	1.50456	dns_query	-0.11304	http_orig_mime	0	dns_query	0.00012	http_status_code	0
36	http_req_body	0.00	http_status_code	0.944575672	http_status_code	1.30068	dns_RA	-0.12804	ssl_subject	0	missed_bytes	0.00012	dns_rejected	0
37	http_resp_body	0.00	weird_notice	0.630217969	http_resp_body	1.2144	dns_qtype	-0.20448	http_req_body	0	src_pkts	0.00012	dns_RD	0
38	http_status_code	0.00	src_pkts	0.379126494	ssl_issuer	1.2	dns_AA	-0.20628	http_trans_depth	0	ssl_issuer	0.00012	dns_rcode	0
39	http_user_agent	0.00	http_user_agent	0.285663105	src_pkts	1.16556	ssl_subject	-0.2946	missed_bytes	0	http_orig_mime	0.00012	dns_RA	0
40	http_orig_mime	0.00	src_ip	0.183319625	http_user_agent	0.41568	ssl_issuer	-0.32184	duration	0	ssl_subject	0.00012	dns_query	0
41	ssl_resumed	0.00	dst_ip_bytes	0.088566405	dst_ip_bytes	0.1776	dst_ip_bytes	-0.3636	src_pkts	0	http_req_body	0.00012	dns_qtype	0
42	weird_addl	0.00	http_resp_mime	0.032715376	http_resp_mime	0.04104	dns_rejected	-0.42888	dst_ip_bytes	0	dst_ip_bytes	0	dns_AA	0
43	http_trans_depth	0.00	duration	0.023533693	src_bytes	0.00012	missed_bytes	-0.51384	http_resp_body	0	http_resp_body	0	weird_addl	0

Table 7.3: Features below the median value for Network_9 Dataset

Feature Rank	IG	Value	χ^2	Value	F-score	Value	R	Value	VT	Value	MAD	Value	DR	Value
1	src_ip	0.84	dns_RD	7433.537	ts	35687.51028	dst_port	0.62472	proto	0.17244	ts	0.36732	ts	1.2011184
2	ts	0.83	ts	5438.695	dst_port	21552.1968	dns_rcode	0.2274	src_port	0.14112	src_port	0.36612	weird_notice	1.2007752
3	src_bytes	0.68	service	2299.514	proto	14053.8498	duration	0.03084	dns_RD	0.14052	conn_state	0.36516	conn_state	1.2002412
4	src_ip_bytes	0.59	src_port	2147.527	src_port	9701.03484	conn_state	0.02904	conn_state	0.1386	dns_RD	0.3228	dst_port	1.2001968
5	conn_state	0.53	dns_rcode	1961.735	dns_AA	7442.97048	ts	0.01188	dns_query	0.10992	service	0.28872	ssl_resumed	1.2001872
6	dst_port	0.52	dns_qtype	1481.856	service	6156.28944	proto	0.0114	dns_rejected	0.10164	proto	0.279	http_version	1.200156
7	duration	0.48	dns_AA	1342.273	dns_rcode	3574.7166	src_ip	0.0114	service	0.0846	dst_pkts	0.2742	src_bytes	1.2001272
8	dst_ip	0.44	proto	1118.175	dst_ip	2923.46592	service	0.00492	ts	0.07368	dns_rejected	0.2718	ssl_version	1.200126
9	src_port	0.41	dst_ip	594.5206	dns_qtype	2870.25624	dst_ip	0.0042	dns_RA	0.07284	dns_RA	0.23568	weird_name	1.20012
10	dst_pkts	0.32	dns_query	533.7794	dns_RA	1104.52248	src_port	0.00168	dns_AA	0.07176	dns_AA	0.23424	dst_ip	1.2000804
11	src_pkts	0.22	dns_RA	521.568	dns_query	859.72968	src_bytes	0	dst_port	0.04116	dst_port	0.1392	http_resp_mime	1.2000732
12	dst_bytes	0.20	dns_qclass	292.4811	conn_state	489.17256	http_resp_mime	-0.00084	dst_ip	0.03384	dst_ip	0.12144	ssl_cipher	1.200066
13	dns_query	0.18	dst_pkts	188.6556	dns_qclass	294.92472	http_user_agent	-0.00252	dst_pkts	0.03084	dns_rcode	0.10044	http_user_agent	1.2000648
14	dst_ip_bytes	0.17	conn_state	125.2734	weird_notice	189.42336	src_pkts	-0.0042	dns_qtype	0.01116	dns_qtype	0.03684	src_port	1.2000648
15	dns_rejected	0.12	weird_addl	80.4	dst_pkts	115.9836	http_resp_body	-0.00432	src_ip	0.01008	src_ip	0.02748	http_method	1.2000408
16	proto	0.11	ssl_resumed	64.88276	ssl_resumed	106.2462	http_status_code	-0.00444	dns_qclass	0.0036	dns_qclass	0.00732	dst_bytes	1.2000276
17	dns_RD	0.09	ssl_established	62.12269	ssl_established	98.12796	dst_pkts	-0.0048	weird_notice	0.00252	weird_notice	0.00504	src_ip_bytes	1.2000144
18	dns_qtype	0.08	http_trans_depth	43.68392	weird_name	84.8274	src_ip_bytes	-0.00588	weird_addl	0.00072	weird_addl	0.00204	dst_pkts	1.2000144
19	service	0.04	weird_name	41.41746	duration	63.6852	http_req_body	-0.0066	ssl_established	0.0006	ssl_established	0.0018	ssl_issuer	1.2000108
20	dns_rcode	0.03	ssl_version	35.26667	dst_bytes	56.1006	http_orig_mime	-0.00708	ssl_resumed	0.00048	ssl_resumed	0.00168	proto	1.2000108
21	dns_AA	0.03	dst_port	28.77967	ssl_version	43.81992	http_uri	-0.00792	http_version	0.00048	weird_name	0.00132	ssl_subject	1.2000096
22	dns_RA	0.02	ssl_cipher	17.87077	dns_RD	42.82896	http_trans_depth	-0.00804	ssl_version	0.00036	src_bytes	0.00108	service	1.2000096

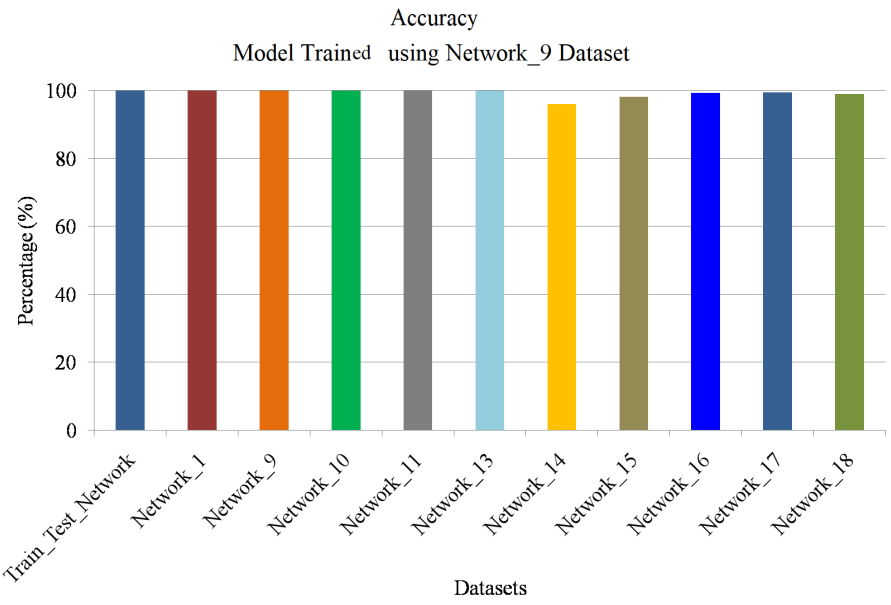


Figure 7.7: RF accuracy after self-healing

Moreover, as evidenced in Figure 7.8, the IDS achieves a recall level above 0.98 in all data files; thus, its capability to detect true positives is almost flawless throughout the data. Additionally, its specificity level achieves 1 in all data files, thus its true negative rate is also flawless. The F1-score achieves a level of 1 in all data files except Network_14 and Network_15; thus, its precision and recall rates are both flawless in all data files by the Random Forest (RF) model. The results obtained suggest that the IDS model is highly efficient in both detecting attack instances (with high recall) and correctly identifying legitimate instances (with perfect specificity levels). The IDS model achieves near-perfect recall values and perfect specificity in all datasets; thus, its efficiency in correctly classifying all instances as malicious or legitimate is highly complemented throughout all datasets.

Figure 7.8: RF performance after self-healing

The retraining time for the IDS model, which uses the Network_9 data file, includes a number of primary sub-tasks such as clustering, ensemble FS, and the retraining of the random forest (RF) model. Each of these sub-tasks plays a critical role in ensuring that the accuracy of the model is achieved. During the retraining time, a total of 74,208 data samples are used with a total time taken of 273 ms. During this time, the retraining process runs at 28% CPU utilization and uses a memory of 1273 MB. After the retraining, the resultant model can then be used in the IoT gateway. The designed IoT-oriented model maintains a storage requirement of 697 KB of storage space, which plays a particularly critical part in storage requirements for IoT devices. The IDS model can therefore be easily transferred to the IoT gateway.

The time required for testing each sample is 0.0050 ms, with 0.1546 MB memory required by the model at CPU utilization of 0.0014%. This shows how efficient the model is with respect to the use of resources and how suitable the model is for use in an environment with limited resources offered by the Internet of things environment. It is efficient since it requires low CPU and Memory.

The resource utilization discussed in this chapter after self-healing is considerably lower than that of the self-healing process without ELIDS discussed in Chapter 6. According to Chapter 6, after self-healing, the memory utilization of the RF model is 3.3 MB, CPU utilization is 0.71%, while time taken to test a sample is 0.47 ms. On the other hand, after incorporating self-healing with ELIDS, a significant reduction is noticed in resource utilization with considerably lower memory and CPU utilization as discussed in this chapter. Such observations prove that the proposed method is highly efficient with no compromise on performance levels.

With the rising number of IoT devices, the efficiency of the intrusion detection system is achieved by the event-driven design and flag-based control of communication, making it scalable. The inter-node communication between the edge device and the sensor/acts as a trigger only when CPU or memory usage exceeds a threshold, implying that the DoS attack has bypassed the edge node. To prevent communication bottlenecks during such attacks, the use of a 'flag' system comes into play: after the activation of the alert by the first IoT device, it sets the 'flag' to '1' to withhold further alerts from the other devices until the completion of the self-healing process of the edge. As it is likely that a standard DDoS attack simultaneously affects the greatest number of IoT devices in real time, a single alert is enough to initiate the

process of mitigation.

7.4 Summary

This chapter discusses an ELIDS-SHIoT capability integration approach, designed specifically to work on resource-challenged IoT systems. The approach follows a multi-step, self-healing mechanism that allows for autonomous systems' detection, response, and recovery capabilities against zero-day attacks. Results show that after the self-healing mechanism, there is a great improvement in the model's accuracy in detecting both known and unknown attacks, achieving an accuracy of 99.9%. Additionally, the model runs in an energy-efficient manner, consuming only 0.1546 MB of memory, 0.0014% CPU usage, and 0.0050 ms on each sample. The retraining time of the ML-IDS is 273 ms in total. All these results identify that the proposed approach not only enhances accuracy in detecting new attack paths but also maintains its efficiency in all aspects of detection, making it an even better candidate to be applied in resource-challenged IoT scenarios. The next chapter summarizes the important outcomes of this study in terms of key findings, discussing aspects of what was achieved and should be further explored in the future.

Bibliography

[1] B. Nagajayanthi, "Decades of internet of things towards twenty-first century: A research-based introspective," *Wireless Personal Communications*, vol. 123, no. 4, pp. 3661–3697, 2022.

[2] N. Palia and D. Kamthania, "Convergence of iot, ai, and wireless technologies in next generation computing," in *Wireless Communication Technologies*. CRC Press, 2024.

[3] S. E. Bibri, J. Krogstie, A. Kaboli, and A. Alahi, "Smarter eco-cities and their leading-edge artificial intelligence of things solutions for environmental sustainability: A comprehensive systematic review," *Environmental Science and Ecotechnology*, vol. 19, p. 100330, 2024.

[4] A. I. Abubakar, H. Chiroma, S. A. Muaz, and L. B. Ila, "A review of the advances in cyber security benchmark datasets for evaluating data-driven based intrusion detection systems," *Procedia Computer Science*, vol. 62, pp. 221–227, 2015.

[5] S. Fischer, "Internet of things: A model for cybersecurity standards and the categorisation of devices," Ph.D. dissertation, 2023.

[6] K. Bhatt, C. Agrawal, and A. M. Bisen, "A review on emerging applications

of iot and sensor technology for industry 4.0," *Wireless Personal Communications*, pp. 1–19, 2024.

[7] A. Khanna and S. Kaur, "Internet of things (iot), applications and challenges: a comprehensive review," *Wireless Personal Communications*, vol. 114, pp. 1687–1762, 2020.

[8] A. Morchid, R. Jebabra, H. M. Khalid, R. El Alami, H. Qjidaa, and M. O. Jamil, "Iot-based smart irrigation management system to enhance agricultural water security using embedded systems, telemetry data, and cloud computing," *Results in Engineering*, vol. 23, p. 102829, 2024.

[9] M. O. Ojo, S. Giordano, G. Procissi, and I. N. Seitanidis, "A review of low-end, middle-end, and high-end iot devices," *IEEE Access*, vol. 6, pp. 70 528–70 554, 2018.

[10] T. Sasi, A. H. Lashkari, R. Lu, P. Xiong, and S. Iqbal, "A comprehensive survey on iot attacks: Taxonomy, detection mechanisms and challenges," *Journal of Information and Intelligence*, 2023.

[11] M. H. Nasir, J. Arshad, and M. M. Khan, "Collaborative device-level botnet detection for internet of things," *Computers & Security*, vol. 129, p. 103172, 2023.

[12] A. Khraisat and A. Alazab, "A critical review of intrusion detection systems in the internet of things: techniques, deployment strategy, validation strategy, attacks, public datasets and challenges," *Cybersecurity*, vol. 4, pp. 1–27, 2021.

[13] M. M. Salim, S. Rathore, and J. H. Park, "Distributed denial of service attacks

and its defenses in iot: a survey," *The Journal of Supercomputing*, vol. 76, pp. 5320–5363, 2020.

[14] A. Aminu Ghali, R. Ahmad, and H. S. A. Alhussian, "Comparative analysis of dos and ddos attacks in internet of things environment," in *Artificial Intelligence and Bioinspired Computational Methods: Proceedings of the 9th Computer Science On-line Conference 2020, Vol. 2 9*. Springer, 2020, pp. 183–194.

[15] S. J. Moore, C. D. Nugent, S. Zhang, and I. Cleland, "Iot reliability: a review leading to 5 key research directions," *CCF Transactions on Pervasive Computing and Interaction*, vol. 2, pp. 147–163, 2020.

[16] X. Zhang, O. Upton, N. L. Beebe, and K.-K. R. Choo, "Iot botnet forensics: A comprehensive digital forensic case study on mirai botnet servers," *Forensic Science International: Digital Investigation*, vol. 32, p. 300926, 2020.

[17] P. Kaliyar, L. Erdodi, and S. Katsikas, "List: Lightweight solutions for securing iot devices against mirai malware attack," in *Proceedings, SECURWARE 2022, The Sixteenth International Conference on Emerging Security Information, Systems and Technologies*. International Academy, Research and Industry Association (IARIA), 2022.

[18] J. Sahota and N. Vlajic, "Mozi iot malware and its botnets: From theory to real-world observations," in *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2021, pp. 698–703.

[19] M. S. Pour, J. Khoury, and E. Bou-Harb, "Honeycomb: A darknet-centric proactive deception technique for curating iot malware forensic artifacts," in

NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium. IEEE, 2022, pp. 1–9.

[20] S. Singh, P. K. Sharma, S. Y. Moon, and J. H. Park, "Advanced lightweight encryption algorithms for iot devices: survey, challenges and solutions," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–18, 2024.

[21] S. Hashemi and M. Zarei, "Internet of things backdoors: Resource management issues, security challenges, and detection methods," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 2, p. e4142, 2021.

[22] M. A. Shyaa, N. F. Ibrahim, Z. Zainol, R. Abdullah, M. Anbar, and L. Alzubaidi, "Evolving cybersecurity frontiers: A comprehensive survey on concept drift and feature dynamics aware machine and deep learning in intrusion detection systems," *Engineering Applications of Artificial Intelligence*, vol. 137, p. 109143, 2024.

[23] G. Kocher and G. Kumar, "Machine learning and deep learning methods for intrusion detection systems: recent developments and challenges," *Soft Computing*, vol. 25, no. 15, pp. 9731–9763, 2021.

[24] A. Thakkar and R. Lohiya, "A survey on intrusion detection system: feature selection, model, performance measures, application perspective, challenges, and future research directions," *Artificial Intelligence Review*, vol. 55, no. 1, pp. 453–563, 2022.

[25] F. M. D. Almeida, A. d. R. L. Ribeiro, and E. D. Moreno, "An architecture for self-healing in internet of things," in *Proceedings of the UBICOMM*, 2015, p. 89.

[26] H. Psaier and S. Dustdar, "A survey on self-healing systems: approaches and systems," *Computing*, vol. 91, pp. 43–73, 2011.

[27] S. Caleb, G. Padmapriya, T. Nandhini, R. Latha *et al.*, "Revolutionizing fault detection in self-healing network via multi-serial cascaded and adaptive network," *Knowledge-Based Systems*, vol. 309, p. 112732, 2025.

[28] S. Kushal, B. Shanmugam, J. Sundaram, and S. Thennadil, "Self-healing hybrid intrusion detection system: an ensemble machine learning approach," *Discover Artificial Intelligence*, vol. 4, no. 1, p. 28, 2024.

[29] G. Guo, X. Pan, H. Liu, F. Li, L. Pei, and K. Hu, "An iot intrusion detection system based on ton iot network dataset," in *2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2023, pp. 0333–0338.

[30] H. Y. I. Khalid and N. B. I. Aldabagh, "A survey on the latest intrusion detection datasets for software defined networking environments," *Engineering, Technology & Applied Science Research*, vol. 14, no. 2, pp. 13 190–13 200, 2024.

[31] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset," *Future Generation Computer Systems*, vol. 100, pp. 779–796, 2019.

[32] A. Mishra, T. Reichherzer, E. Kalaimannan, N. Wilde, and R. Ramirez, "Trade-offs involved in the choice of cloud service configurations when building secure, scalable, and efficient internet-of-things networks," *International*

Journal of Distributed Sensor Networks, vol. 16, no. 2, p. 1550147720908199, 2020.

[33] M. Humayun, N. Tariq, M. Alfayad, M. Zakwan, G. Alwakid, and M. Assiri, "Securing the internet of things in artificial intelligence era: A comprehensive survey," *IEEE access*, vol. 12, pp. 25 469–25 490, 2024.

[34] Imperva. (2017) Counterbreach 2.0 introduces new machine learning algorithm to protect data against insider threats.

[35] P. Kumari and A. K. Jain, "A comprehensive study of ddos attacks over iot network and their countermeasures," *Computers & Security*, vol. 127, p. 103096, 2023.

[36] D. M. Sharif, H. Beitollahi, and M. Fazeli, "Detection of application-layer ddos attacks produced by various freely accessible toolkits using machine learning," *IEEE Access*, vol. 11, pp. 51 810–51 819, 2023.

[37] M. M. Khan, M. Murphy, and A. Beige, "High error-rate quantum key distribution for long-distance communication," *New Journal of Physics*, vol. 11, no. 6, p. 063043, 2009.

[38] M. Fatima, O. Rehman, and I. M. Rehman, "Li-IDS: An approach towards a lightweight ids for resource-constrained iot," in *2023 International Conference on Smart Applications, Communications and Networking (SmartNets)*. IEEE, 2023, pp. 1–6.

[39] C. C. Center, "Denial of service attacks," Carnegie Mellon University, Tech. Rep., 1999, available: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=498084.

[40] D. Moore, G. M. Voelker, and S. Savage, "Inferring internet denial-of-service activity," in *Proceedings of the 10th USENIX Security Symposium*, 2001, pp. 9–22.

[41] Z. Zhang, G. Xiao, S. Song, R. C. Aygun, A. Stavrou, L. Zhang, and E. Osterweil, "Revealing protocol architecture's design patterns in the volumetric ddos defense design space," *IEEE Communications Surveys & Tutorials*, 2024.

[42] H. Wang, D. Zhang, and K. G. Shin, "Detecting syn flooding attacks," in *Proceedings. Twenty-first annual joint conference of the IEEE computer and communications societies*, vol. 3. IEEE, 2002, pp. 1530–1539.

[43] S. Scheuer, D. Haase, and V. Meyer, "Exploring multicriteria flood vulnerability by integrating economic, social and ecological dimensions of flood risk and coping capacity: from a starting point view towards an end point view of vulnerability," *Natural hazards*, vol. 58, pp. 731–751, 2011.

[44] A. Lavrenovs, "Measuring distributed reflected denial-of-service amplified volumetric attack capacity: Doctoral thesis," 2023.

[45] E. Damon, J. Dale, E. Laron, J. Mache, N. Land, and R. Weiss, "Hands-on denial of service lab exercises using slowloris and rudy," in *proceedings of the 2012 information security curriculum development conference*, 2012, pp. 21–29.

[46] A. G. Eustis, "The mirai botnet and the importance of iot device security," in *16th International Conference on Information Technology-New Generations (ITNG 2019)*, 2019, pp. 85–89.

[47] M. Pelloso, A. Vergutz, A. Santos, and M. Nogueira, "A self-adaptable system

for ddos attack prediction based on the metastability theory," in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–6.

[48] A. Asokan. (2019, Jul) Massive botnet attack used more than 400,000 iot devices. [Online]. Available: https://www.bankinfosecurity.com/massive-botnet-attack-used-more-than-400000-iot-devices-a-12841

[49] A. W. Services, "Amazon web services faces ddos attack reaching 2.3 tbps in february 2020," *AWS Security Blog*, 2020, accessed: 2025-04-04. [Online]. Available: https://aws.amazon.com/blogs/security/

[50] M. Azure, "Azure faces peak ddos attack staggering at 3.47 tbps," *Microsoft Azure Blog*, 2021, accessed: 2025-04-04. [Online]. Available: https://techcommunity.microsoft.com/t5/azure-security/

[51] Cloudflare, "Cloudflare faces record-breaking ddos attack, peaking at 5.6 tbps," *Cloudflare Blog*, 2024, accessed: 2025-04-04. [Online]. Available: https://www.cloudflare.com/learning/ddos/

[52] O. H. Abdulganiyu, T. Ait Tchakoucht, and Y. K. Saheed, "A systematic literature review for network intrusion detection system (ids)," *International journal of information security*, vol. 22, no. 5, pp. 1125–1162, 2023.

[53] Q. Chi, H. Yan, C. Zhang, Z. Pang, and L. D. Xu, "A reconfigurable smart sensor interface for industrial wsn in iot environment," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1417–1425, 2014.

[54] A. V. Joshi, "Machine learning and artificial intelligence," 2020.

[55] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[56] H. Nozari, J. Ghahremani-Nahr, and A. Szmelter-Jarosz, "Ai and machine learning for real-world problems," in *Advances In Computers*. Elsevier, 2024, vol. 134, pp. 1–12.

[57] B. Mahesh, "Machine learning algorithms-a review," *International Journal of Science and Research (IJSR).[Internet]*, vol. 9, no. 1, pp. 381–386, 2020.

[58] L. Photonics, "nonlinear interactions of laser light and matter, r. menzel," 2001.

[59] Y.-Y. Song and L. Ying, "Decision tree methods: applications for classification and prediction," *Shanghai archives of psychiatry*, vol. 27, no. 2, p. 130, 2015.

[60] D. Rani and N. C. Kaushal, "Supervised machine learning based network intrusion detection system for internet of things," in *2020 11th International conference on computing, communication and networking technologies (ICCCNT)*. IEEE, 2020, pp. 1–7.

[61] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," *Frontiers in neurorobotics*, vol. 7, p. 21, 2013.

[62] E. Anthi, L. Williams, M. Słowińska, G. Theodorakopoulos, and P. Burnap, "A supervised intrusion detection system for smart home iot devices," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 9042–9053, 2019.

[63] G. I. Webb, E. Keogh, and R. Miikkulainen, "Naïve bayes." *Encyclopedia of machine learning*, vol. 15, no. 1, pp. 713–714, 2010.

[64] D. Sujatha, M. R. TF, G. Ramesh, M. Agoramoorthy *et al.*, "Neural networks-based predictive models for self-healing in cloud computing environments," in

2024 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE). IEEE, 2024, pp. 1–4.

[65] M. H. Kabir, M. S. Rajib, A. S. M. T. Rahman, M. M. Rahman, and S. K. Dey, "Network intrusion detection using unsw-nb15 dataset: Stacking machine learning based approach," in *2022 International Conference on Advancement in Electrical and Electronic Engineering (ICAEEE)*. IEEE, 2022, pp. 1–6.

[66] T. M. Kodinariya, P. R. Makwana *et al.*, "Review on determining number of cluster in k-means clustering," *International Journal*, vol. 1, no. 6, pp. 90–95, 2013.

[67] M. A. Nielsen and I. L. Chuang, "Quantum computation and quantum information," *Phys. Today*, vol. 54, no. 2, p. 60, 2001.

[68] M. M. Khan, J. Xu, and A. Beige, "A detailed analysis of kmb09 qkd protocol," *International Journal of Computer Science and Information Security*, vol. 15, no. 1, p. 529, 2017.

[69] G. J. McLachlan and S. Rathnayake, "On the number of components in a gaussian mixture model," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 4, no. 5, pp. 341–355, 2014.

[70] A. Maćkiewicz and W. Ratajczak, "Principal components analysis (pca)," *Computers & Geosciences*, vol. 19, no. 3, pp. 303–342, 1993.

[71] M. C. Cieslak, A. M. Castelfranco, V. Roncalli, P. H. Lenz, and D. K. Hartline, "t-distributed stochastic neighbor embedding (t-sne): A tool for eco-physiological transcriptomic analysis," *Marine genomics*, vol. 51, p. 100723, 2020.

[72] M. Tschannen, O. Bachem, and M. Lucic, "Recent advances in autoencoder-based representation learning," *arXiv preprint arXiv:1812.05069*, 2018.

[73] J. Clifton and E. Laber, "Q-learning: Theory and applications," *Annual Review of Statistics and Its Application*, vol. 7, no. 1, pp. 279–301, 2020.

[74] D. Zhao, H. Wang, K. Shao, and Y. Zhu, "Deep reinforcement learning with experience replay based on sarsa," in *2016 IEEE symposium series on computational intelligence (SSCI)*. IEEE, 2016, pp. 1–6.

[75] M. Roderick, J. MacGlashan, and S. Tellex, "Implementing the deep q-network," *arXiv preprint arXiv:1711.07478*, 2017.

[76] B. T. Polyak, "The conjugate gradient method in extremal problems," *USSR Computational Mathematics and Mathematical Physics*, vol. 9, no. 4, pp. 94–112, 1969.

[77] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.

[78] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[79] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[80] M. Sewak and M. Sewak, "Actor-critic models and the a3c: The asynchronous advantage actor-critic model," *Deep reinforcement learning: frontiers of artificial intelligence*, pp. 141–152, 2019.

[81] D. Dhiman, A. Bisht, G. Thakur, and A. Garg, "Artificial intelligence and machine learning-enabled cybersecurity tools and techniques," in *Advanced Techniques and Applications of Cybersecurity and Forensics*. Chapman and Hall/CRC, 2025, pp. 35–56.

[82] S. Santhosh Kumar, M. Selvi, and A. Kannan, "A comprehensive survey on machine learning-based intrusion detection systems for secure communication in internet of things," *Computational Intelligence and Neuroscience*, vol. 2023, no. 1, p. 8981988, 2023.

[83] K. He, D. D. Kim, and M. R. Asghar, "Adversarial machine learning for network intrusion detection systems: A comprehensive survey," *IEEE Communications Surveys and Tutorials*, vol. 25, no. 1, pp. 538–566, 2023.

[84] F. Kühn, H. Hellbrück, and S. Fischer, "A model-based approach for self-healing iot systems," in *Proceedings of the 7th International Conference on Sensor Networks*, 2018, pp. 135–140.

[85] Z. Zoghi and G. Serpen, "Unsw-nb15 computer security dataset: Analysis through visualization," *Security and Privacy*, vol. 7, no. 1, p. e331, 2024.

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

[87] D. D. Protić, "Review of kdd cup '99, nsl-kdd and kyoto 2006+ datasets," *Vojnotehnički glasnik/Military Technical Courier*, vol. 66, no. 3, pp. 580–596, 2018.

[88] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network

intrusion detection systems (unsw-nb15 network data set)," in *2015 military communications and information systems conference (MilCIS)*. IEEE, 2015, pp. 1–6.

[89] M. S. Korium, M. Saber, A. Beattie, A. Narayanan, S. Sahoo, and P. H. Nardelli, "Intrusion detection system for cyberattacks in the internet of vehicles environment," *Ad Hoc Networks*, vol. 153, p. 103330, 2024.

[90] I. Almomani, B. Al-Kasasbeh, and M. Al-Akhras, "Wsn-ds: a dataset for intrusion detection systems in wireless sensor networks," *Journal of Sensors*, vol. 2016, no. 1, p. 4731953, 2016.

[91] M. Xie and J. Hu, "Evaluating host-based anomaly detection systems: A preliminary analysis of adfa-ld," in *2013 6th international congress on image and signal processing (CISP)*, vol. 3. IEEE, 2013, pp. 1711–1716.

[92] J. M. Peterson, J. L. Leevy, and T. M. Khoshgoftaar, "A review and analysis of the bot-iot dataset," in *2021 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, 2021, pp. 20–27.

[93] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, and A. Anwar, "Ton_iot telemetry dataset: A new generation dataset of iot and iiot for data-driven intrusion detection systems," *Ieee Access*, vol. 8, pp. 165 130–165 150, 2020.

[94] P. Velan, M. Čermák, P. Čeleda, and M. Drašar, "A survey of methods for encrypted traffic classification and analysis," *International Journal of Network Management*, vol. 25, no. 5, pp. 355–374, 2015.

[95] K. N. Junejo and J. Goh, "Behaviour-based attack detection and classifica-

tion in cyber physical systems using machine learning," in *Proceedings of the [Conference Name].*

[96] M. Pal and G. M. Foody, "Feature selection for classification of hyperspectral data by svm," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 48, no. 5, pp. 2297–2307, 2010.

[97] J. Cai, J. Luo, S. Wang, and S. Yang, "Feature selection in machine learning: A new perspective," *Neurocomputing*, vol. 300, pp. 70–79, 2018.

[98] C. W. Chen, Y. H. Tsai, F. R. Chang, and W. C. Lin, "Ensemble feature selection in medical datasets: Combining filter, wrapper, and embedded feature selection results," *Expert Systems*, vol. 37, no. 5, p. e12553, 2020.

[99] E. O. Omuya, G. O. Okeyo, and M. W. Kimwele, "Feature selection for classification using principal component analysis and information gain," *Expert Systems with Applications*, vol. 174, p. 114765, 2021.

[100] I. S. Thaseen and C. A. Kumar, "Intrusion detection model using fusion of chi-square feature selection and multi class svm," *Journal of King Saud University-Computer and Information Sciences*, vol. 29, no. 4, pp. 462–472, 2017.

[101] I. Visentini, L. Snidaro, and G. L. Foresti, "Diversity-aware classifier ensemble selection via f-score," *Information Fusion*, vol. 28, pp. 24–43, 2016.

[102] Y. Liu, Y. Mu, K. Chen, Y. Li, and J. Guo, "Daily activity feature selection in smart homes based on pearson correlation coefficient," *Neural Processing Letters*, vol. 51, pp. 1771–1787, 2020.

[103] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[104] Y. Simaan, "Estimation risk in portfolio selection: the mean variance model versus the mean absolute deviation model," *Management science*, vol. 43, no. 10, pp. 1437–1446, 1997.

[105] T. A. Alamiedy, M. Anbar, A. K. Al-Ani, B. N. Al-Tamimi, and N. Faleh, "Review on feature selection algorithms for anomaly-based intrusion detection system," in *Proceedings of the [Conference Name]*.

[106] T. H. Hai, E. N. Huh, and M. Jo, "A lightweight intrusion detection framework for wireless sensor networks," *Wireless Communications and Mobile Computing*, vol. 10, no. 4, pp. 559–572, 2010.

[107] Y. Maleh and A. Ezzati, "Lightweight intrusion detection scheme for wireless sensor networks," *IAENG International Journal of Computer Science*, vol. 42, no. 4, 2015.

[108] M. Roesch, "Snort: Lightweight intrusion detection for networks," in *Proceedings of the 13th USENIX Conference on System Administration (LISA '99)*. USENIX Association, 1999, pp. 229–238.

[109] M. Othman, S. A. Madani, and S. U. Khan, "A survey of mobile cloud computing application models," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 393–413, 2013.

[110] S. A. Khanday, H. Fatima, and N. Rakesh, "Implementation of intrusion detection model for ddos attacks in lightweight iot networks," *Expert Systems with Applications*, vol. 215, p. 119330, 2023.

[111] J. Azimjonov and T. Kim, "Designing accurate lightweight intrusion detection

systems for iot networks using fine-tuned linear svm and feature selectors," *Computers & Security*, vol. 137, p. 103598, 2024.

[112] Y. K. Saheed, A. I. Abiodun, S. Misra, M. K. Holone, and R. Colomo-Palacios, "A machine learning-based intrusion detection for detecting internet of things network attacks," *Alexandria Engineering Journal*, vol. 61, no. 12, pp. 9395–9409, 2022.

[113] S. Roy, J. Li, B.-J. Choi, and Y. Bai, "A lightweight supervised intrusion detection mechanism for iot networks," *Future Generation Computer Systems*, vol. 127, pp. 276–285, 2022.

[114] K. M. Sai, B. B. Gupta, C.-H. Hsu, and D. Peraković, "Lightweight intrusion detection system in iot networks using raspberry pi 3b+." in *SysCom*, 2021, pp. 43–51.

[115] E. Özer, M. Iskefiyeli, and J. Azimjonov, "Toward lightweight intrusion detection systems using the optimal and efficient feature pairs of the bot-iot 2018 dataset," *International Journal of Distributed Sensor Networks*, vol. 17, no. 10, p. 15501477211052202, 2021.

[116] M. Omar and L. George, "Toward a lightweight machine learning based solution against cyber-intrusions for iot," in *2021 IEEE 46th Conference on Local Computer Networks (LCN)*. IEEE, 2021, pp. 519–524.

[117] B. S. Khater, A. W. Abdul Wahab, M. Y. I. Idris, M. A. Hussain, A. A. Ibrahim, M. A. Amin, and H. A. Shehadeh, "Classifier performance evaluation for lightweight ids using fog computing in iot security," *Electronics*, vol. 10, no. 14, p. 1633, 2021.

[118] S. J. Lee, P. D. Yoo, A. T. Asyhari, Y. Jhi, L. Chermak, C. Y. Yeun, and K. Taha, "Impact: Impersonation attack detection via edge computing using deep autoencoder and feature abstraction," *IEEE Access*, vol. 8, pp. 65 520–65 529, 2020.

[119] V. Kumar, D. Sinha, A. K. Das, S. C. Pandey, and R. T. Goswami, "An integrated rule based intrusion detection system: analysis on unsw-nb15 data set and the real time online dataset," *Cluster Computing*, vol. 23, pp. 1397–1418, 2020.

[120] N. A. Hikal and M. Elgayar, "Enhancing iot botnets attack detection using machine learning-ids and ensemble data preprocessing technique," in *Internet of Things—Applications and Future: Proceedings of ITAF 2019*. Springer, 2020, pp. 89–102.

[121] S. Ahn, H. Yi, Y. Lee, W. R. Ha, G. Kim, and Y. Paek, "Hawkware: Network intrusion detection based on behavior analysis with anns on an iot device," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[122] J. Harriman, S. Serati, and J. Stockley, "Comparison of transmissive and reflective spatial light modulators for optical manipulation applications," in *Optical Trapping and Optical Micromanipulation II*, vol. 5930. International Society for Optics and Photonics, 2005, p. 59302D.

[123] F. A. Bakhtiar, E. S. Pramukantoro, and H. Nihri, "A lightweight ids based on j48 algorithm for detecting dos attacks on iot middleware," in *2019 IEEE 1st*

global conference on life sciences and technologies (LifeTech). IEEE, 2019, pp. 41–42.

[124] S. U. Jan, S. Ahmed, V. Shakhov, and I. Koo, "Toward a lightweight intrusion detection system for the internet of things," *IEEE access*, vol. 7, pp. 42 450–42 471, 2019.

[125] C. Kavitha, T. R. Gadekallu, N. K. N, B. P. Kavin, and W.-C. Lai, "Filter-based ensemble feature selection and deep learning model for intrusion detection in cloud computing," *Electronics*, vol. 12, no. 3, p. 556, 2023.

[126] Y. Zhang, H. Zhang, and B. Zhang, "An effective ensemble automatic feature selection method for network intrusion detection," *Information*, vol. 13, no. 7, p. 314, 2022.

[127] F. Alserhani and A. Aljared, "Evaluating ensemble learning mechanisms for predicting advanced cyber attacks," *Applied Sciences*, vol. 13, no. 24, p. 13310, 2023.

[128] Y. Zhang, H. Zhang, and B. Zhang, "An effective ensemble automatic feature selection method for network intrusion detection," *Information*, vol. 13, no. 7, p. 314, 2022.

[129] D. P. M. Abellana, R. R. Roxas, D. M. Lao, P. E. Mayol, and S. Lee, "Ensemble feature selection in binary machine learning classification: A novel application of the evaluation based on distance from average solution (edas) method," *Mathematical Problems in Engineering*, vol. 2022, no. 1, p. 4126536, 2022.

[130] N. Hoque, M. Singh, and D. K. Bhattacharyya, "Efs-mi: an ensemble feature

selection method for classification: An ensemble feature selection method," *Complex Intelligent Systems*, vol. 4, pp. 105–118, 2018.

[131] K. Albulayhi, Q. Abu Al-Haija, S. A. Alsuhibany, A. A. Jillepalli, M. Ashrafuzzaman, and F. T. Sheldon, "Iot intrusion detection using machine learning with a novel high performing feature selection method," *Applied Sciences*, vol. 12, no. 10, p. 5015, 2022.

[132] I. Ullah and Q. H. Mahmoud, "A scheme for generating a dataset for anomalous activity detection in iot networks," pp. 508–520.

[133] A. Alhowaide, I. Alsmadi, and J. Tang, "An ensemble feature selection method for iot ids," pp. 41–48.

[134] J. Arshad, M. A. Azad, M. M. Abdeltaif, and K. Salah, "An intrusion detection framework for energy constrained iot devices," *Mechanical Systems and Signal Processing*, vol. 136, p. 106436, 2020.

[135] T. Shao, D. Chowdhury, S. S. Gill, and R. Buyya, "Iot-pi: a machine learning-based lightweight framework for cost-effective distributed computing using iot," *Internet Technology Letters*, vol. 5, no. 3, p. e355, 2022.

[136] Y. Labiod, A. Amara Korba, and N. Ghoualmi, "Fog computing-based intrusion detection architecture to protect iot networks," *Wireless Personal Communications*, vol. 125, no. 1, pp. 231–259, 2022.

[137] A. Mudgerikar, P. Sharma, and E. Bertino, "Edge-based intrusion detection for iot devices," *ACM Transactions on Management Information Systems (TMIS)*, vol. 11, no. 4, pp. 1–21, 2020.

[138] H. Ap and K. K, "Secure-mqtt: an efficient fuzzy logic-based approach to

detect dos attack in mqtt protocol for internet of things," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, p. 90, 2019.

[139] Y. N. Soe, Y. Feng, P. I. Santosa, R. Hartanto, and K. Sakurai, "Implementing lightweight iot-ids on raspberry pi using correlation-based feature selection and its performance evaluation," in *Advanced Information Networking and Applications: Proceedings of the 33rd International Conference on Advanced Information Networking and Applications (AINA-2019) 33*. Springer, 2020, pp. 458–469.

[140] Z. Chen and J. Dongarra, "Highly scalable self-healing algorithms for high performance scientific computing," *IEEE Transactions on Computers*, vol. 58, no. 11, pp. 1512–1524, 2009.

[141] C. Blundo, P. D'arco, A. De Santis, and M. Listo, "Design of self-healing key distribution schemes," *Designs, Codes and Cryptography*, vol. 32, pp. 15–44, 2004.

[142] G. Amit, A. Shabtai, and Y. Elovici, "A self-healing mechanism for internet of things devices," *IEEE Security & Privacy*, vol. 19, no. 1, pp. 44–53, 2020.

[143] O. Johnphill, A. S. Sadiq, F. Al-Obeidat, H. Al-Khateeb, M. A. Taheir, O. Kaiwartya, and M. Ali, "Self-healing in cyber–physical systems using machine learning: A critical analysis of theories and tools," *Future Internet*, vol. 15, no. 7, p. 244, 2023.

[144] V. Degeler, R. French, and K. Jones, "Self-healing intrusion detection system concept," in *2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Perfor-*

mance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS). IEEE, 2016, pp. 351–356.

[145] X. Liu, G. Su, Q. Guo, C. Lu, T. Zhou, C. Zhou, and X. Zhang, "Hierarchically structured self-healing sensors with tunable positive/negative piezoresistivity," *Advanced Functional Materials*, vol. 28, no. 15, p. 1706658, 2018.

[146] C. Keliris and M. M. Polycarpou, "Fault diagnosis in cyber-physical systems," in *Encyclopedia of Cryptography, Security and Privacy*. Springer, 2021, pp. 1–5.

[147] F. M. de Almeida, A. d. R. L. Ribeiro, and E. D. Moreno, "An architecture for self-healing in internet of things," in *Proceedings of the UBICOMM*, vol. 89, 2015, pp. 76–81.

[148] P. Murala, K. Nayak, K. V. Kumar, and S. Vishwakarma, "Bio-inspired algorithms for self-healing and adaptation in sensor networks," in *2024 IEEE 13th International Conference on Communication Systems and Network Technologies (CSNT)*. IEEE, 2024, pp. 35–41.

[149] S. N. Swamy, D. Jadhav, and N. Kulkarni, "Security threats in the application layer in iot applications," in *2017 International Conference on i-SMAC (IoT in Social, Mobile, Analytics and Cloud) (i-SMAC)*. IEEE, 2017, pp. 3–7.

[150] J. Schwinger, "The special canonical group," *Proceedings of the national academy of sciences of the United States of America*, vol. 46, no. 10, p. 1401, 1960.

[151] D. Bacco, J. B. Christensen, M. A. U. Castaneda, Y. Ding, S. Forchhammer, K. Rottwitt, and L. K. Oxenløwe, "Two-dimensional distributed-phase-

reference protocol for quantum key distribution," *Scientific reports*, vol. 6, no. 1, pp. 1–7, 2016.

[152] V. K. Singh, E. Vaughan, and J. Rivera, "Sharp-net: Platform for self-healing and attack resilient pmu networks," in *Proceedings of the Energy Security and Resilience Center, National Renewable Energy Laboratory.* National Renewable Energy Laboratory, 2024. [Online]. Available: https://example-link-to-paper.org

[153] G. S. Almeida and F. E. Vasconcelos, "Self-healing networks: Adaptive responses to ransomware attacks," *Preprints.org*, 2023, not peer-reviewed version. [Online]. Available: https://doi.org/10.20944/preprints202312.1538.v1

[154] B. Abdulrazak, J. A. Codjo, and S. Paul, "Self-healing approach for iot architecture: Ami platform," in *International Conference on Smart Living and Public Health (ICOST).* Springer, 2022, pp. 3–17. [Online]. Available: https://doi.org/10.1007/978-3-031-09593-1_1

[155] J. D. Pineda, G. Diocampo, J. M. P. Hipolito, M. I. Jovellano, and M. F. Pachico, "Implementing an agent-based monitoring approach towards self-healing networks," *Asia Pacific College*, 2017. [Online]. Available: https://www.apc.edu.ph

[156] R. Seiger, S. Herrmann, and U. Aßmann, "Self-healing for distributed workflows in the internet of things," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW).* IEEE, 2017, pp. 72–79.

[157] A. M. Alhomoud, I. U. Awan, and J. F. P. Disso, "Towards an enterprise self-healing system against botnet attacks," in *2013 International Conference on Computing,*

Networking and Communications (ICNC). IEEE, 2013, pp. 1112–1117. [Online]. Available: https://ieeexplore.ieee.org/document/6504248

[158] P. Jain, P. K. Singh, and A. Abraham, "Intrusion detection and self-healing model for network security," in *2011 7th International Conference on Next Generation Web Services Practices*. IEEE, 2011, pp. 320–325. [Online]. Available: https://ieeexplore.ieee.org/document/6088181

[159] 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.

[160] Z. K. Maseer, R. Yusof, N. Bahaman, S. A. Mostafa, and C. F. M. Foozy, "Benchmarking of machine learning for anomaly based intrusion detection systems in the cicids2017 dataset," *IEEE access*, vol. 9, pp. 22 351–22 370, 2021.

[161] G. Rodola, "Psutil documentation," *Psutil. https://psutil. readthedocs. io/en/latest*, 2020.

[162] O. Dor and Y. Zhou, "Achieving 80% ten-fold cross-validated accuracy for secondary structure prediction by large-scale training," *Proteins: Structure, Function, and Bioinformatics*, vol. 66, no. 4, pp. 838–845, 2007.

[163] M. Azzeh, Y. Alqasrawi, and Y. Elsheikh, "A soft computing approach for software defect density prediction," *Journal of Software: Evolution and Process*, vol. 36, no. 4, p. e2553, 2024.

[164] J. Zhang, S. Li, H. Yang, J. Jiang, and H. Shi, "Efficient and intelligent feature selection via maximum conditional mutual information for microarray data." *Applied Sciences (2076-3417)*, vol. 14, no. 13, 2024.

[165] Y. Karaca and C. Cattani, *Computational methods for data analysis*. Walter de Gruyter GmbH & Co KG, 2018.

[166] D. Ghosh, R. Sharman, H. R. Rao, and S. Upadhyaya, "Self-healing systems—survey and synthesis," *Decision support systems*, vol. 42, no. 4, pp. 2164–2185, 2007.

[167] H. Sorg, D. J. Tilkorn, S. Hager, J. Hauser, and U. Mirastschijski, "Skin wound healing: an update on the current knowledge and concepts," *European surgical research*, vol. 58, no. 1-2, pp. 81–94, 2017.

[168] K. Khalil, O. Eldash, A. Kumar, and M. Bayoumi, "Self-healing hardware systems: A review," *Microelectronics Journal*, vol. 93, p. 104620, 2019.

[169] S. S. Gill, H. Wu, P. Patros, C. Ottaviani, P. Arora, V. C. Pujol, D. Haunschild, A. K. Parlikad, O. Cetinkaya, H. Lutfiyya *et al.*, "Modern computing: Vision and challenges," *Telematics and Informatics Reports*, p. 100116, 2024.

[170] M. M. Al-Zawi, D. Al-Jumeily, A. Hussain, and A. Taleb-Bendiab, "Autonomic computing: Applications of self-healing systems," in *2011 Developments in E-systems Engineering*, 2011, pp. 381–386.

[171] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[172] D. S. Punithavathani, K. Sujatha, and J. M. Jain, "Surveillance of anomaly and misuse in critical networks to counter insider threats using computational intelligence," *Cluster Computing*, vol. 18, pp. 435–451, 2015.

[173] A. Ahda, C. Wulandari, H. P. Husellvi, M. Y. Alhuda, M. Reda, P. Zahwa, and

S. Ananda, "Information security implementation of ddos attack using hping3 tools," *JComce-Journal of Computer Science*, vol. 1, no. 4, 2023.

[174] F. Holik, J. Horalek, O. Marik, S. Neradova, and S. Zitta, "Effective penetration testing with metasploit framework and methodologies," in *2014 IEEE 15th International Symposium on Computational Intelligence and Informatics (CINTI)*. IEEE, 2014, pp. 237–242.

[175] M. Sauter, ""loic will tear us apart" the impact of tool design and media portrayals in the success of activist ddos attacks," *American Behavioral Scientist*, vol. 57, no. 7, pp. 983–1007, 2013.

[176] R. L. Keeney and D. Von Winterfeldt, "A value model for evaluating homeland security decisions," *Risk Analysis: An International Journal*, vol. 31, no. 9, pp. 1470–1487, 2011.

[177] D.-j. Kim, B. A. NG, and V. Sarveshwaran, "A novel split learning based consumer electronics network traffic anomaly detection framework for smart city environment," *IEEE Transactions on Consumer Electronics*, 2024.

[178] X. Zhang, R. Zhao, Z. Jiang, Z. Sun, Y. Ding, E. C. Ngai, and S.-H. Yang, "Aoc-ids: Autonomous online framework with contrastive learning for intrusion detection," *arXiv preprint arXiv:2402.01807*, 2024.

[179] C. Constantinides, S. Shiaeles, B. Ghita, and N. Kolokotronis, "A novel online incremental learning intrusion prevention system," in *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE, 2019, pp. 1–6.

[180] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern recognition*, vol. 36, no. 2, pp. 451–461, 2003.

[181] A. M. Ikotun, A. E. Ezugwu, L. Abualigah, B. Abuhaija, and J. Heming, "K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data," *Information Sciences*, vol. 622, pp. 178–210, 2023.

[182] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable k-means++," *arXiv preprint arXiv:1203.6402*, 2012.

[183] S. Zahra, M. A. Ghazanfar, A. Khalid, M. A. Azam, U. Naeem, and A. Prugel-Bennett, "Novel centroid selection approaches for kmeans-clustering based recommender systems," *Information sciences*, vol. 320, pp. 156–189, 2015.

[184] M. Li, E. Frank, and B. Pfahringer, "Large scale k-means clustering using gpus," *Data Mining and Knowledge Discovery*, vol. 37, no. 1, pp. 67–109, 2023.

[185] M. Capó, A. Pérez, and J. A. Lozano, "A cheap feature selection approach for the k-means algorithm," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 5, pp. 2195–2208, 2020.

[186] M. Zounemat-Kermani, O. Batelaan, M. Fadaee, and R. Hinkelmann, "Ensemble machine learning paradigms in hydrology: A review," *Journal of Hydrology*, vol. 598, p. 126266, 2021.

[187] U. Bilal, "Understanding neural networks: A comprehensive overview of ai techniques," *Frontiers in Artificial Intelligence Research*, vol. 1, no. 02, pp. 172–208, 2024.

[188] UNSW, "Toniot datasets," https://research.unsw.edu.au/projects/toniot-datasets, 2024, accessed: 2024-11-06.

Appendix A
Random Forest Built using TON_IoT (Train_Test_Network) Dataset

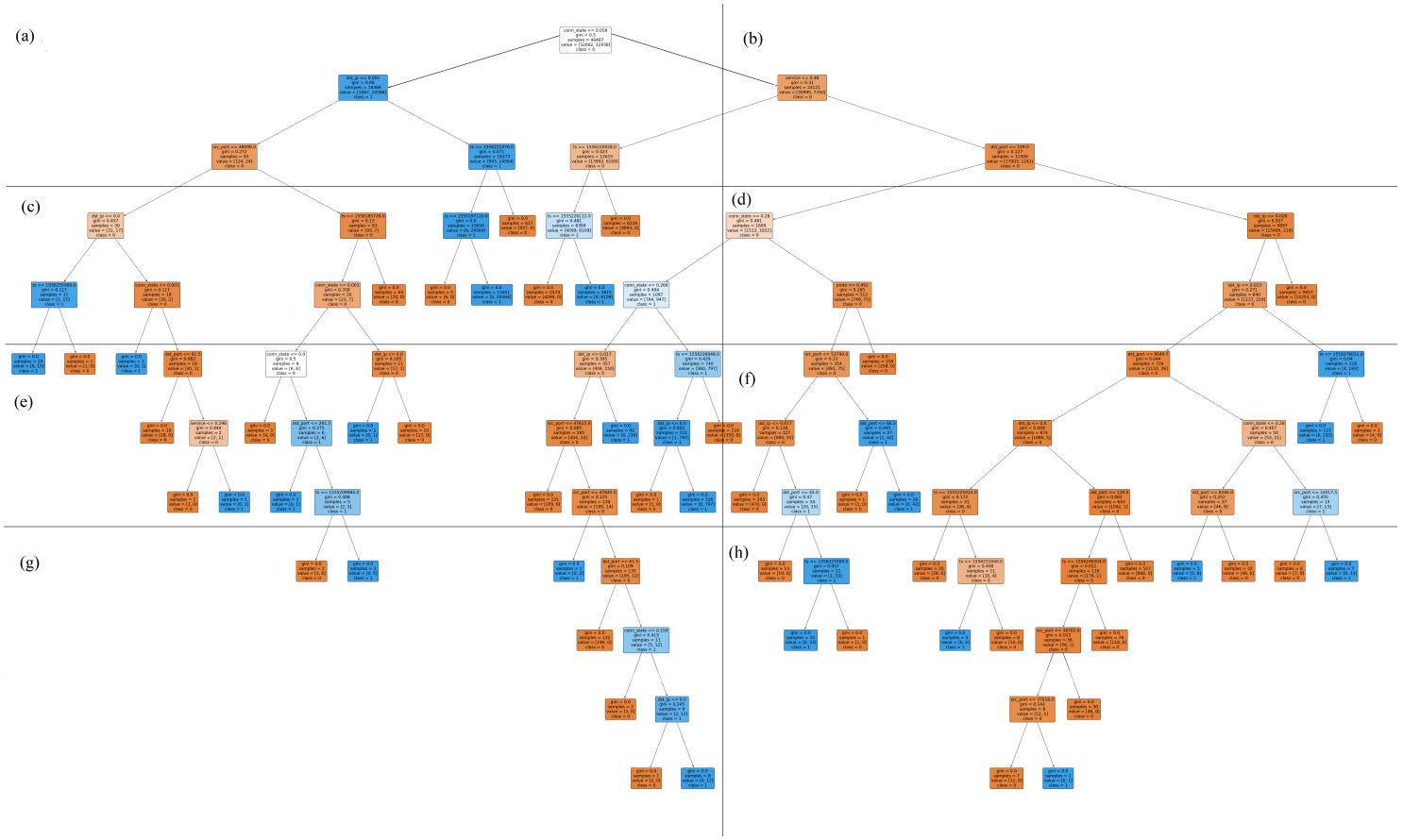


Figure A.1: Random Forest built using Train_Test_Network file

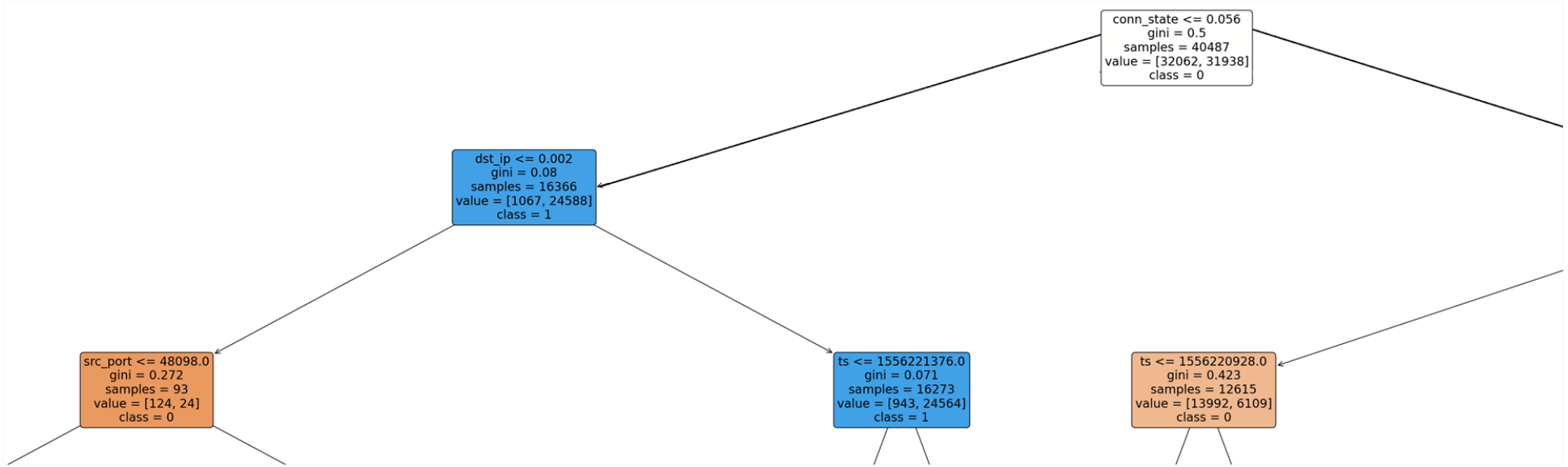


Figure A.2: Random Forest built using Train_Test_Network: part (a)

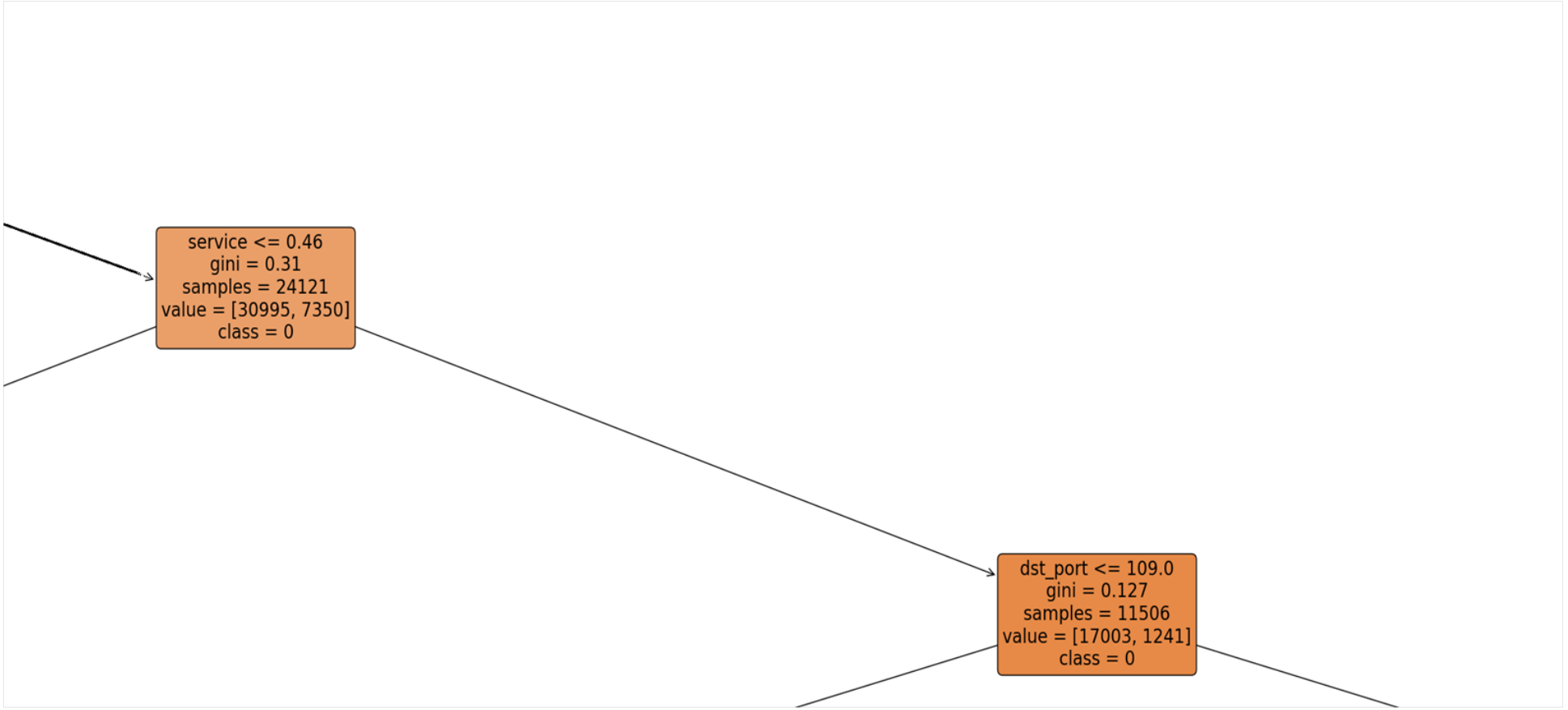


Figure A.3: Random Forest built using Train_Test_Network: part (b)

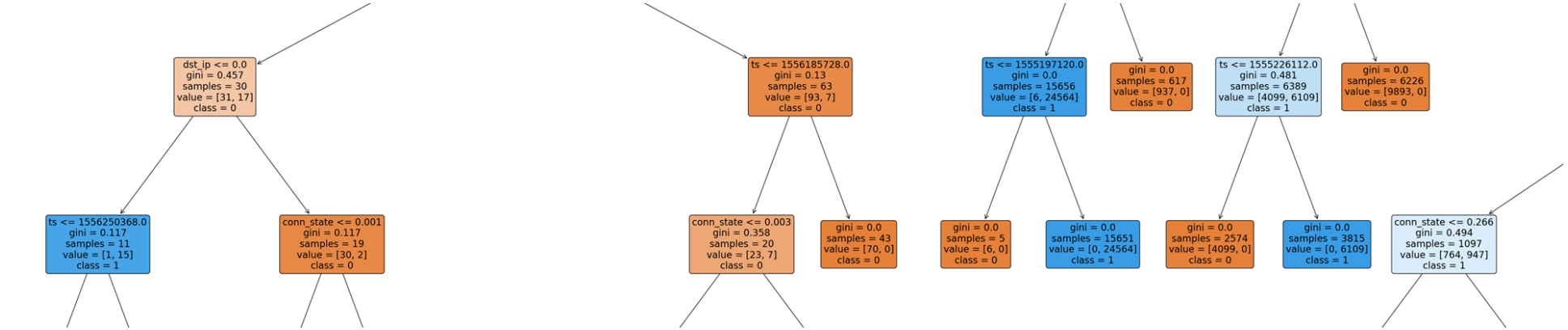


Figure A.4: Random Forest built using Train_Test_Network: part (c)

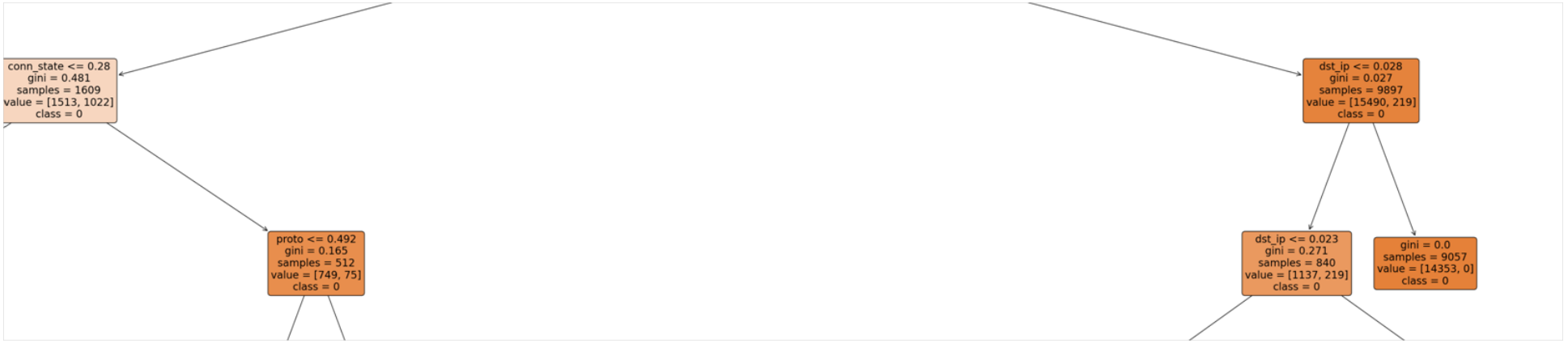


Figure A.5: Random Forest built using Train_Test_Network: part (d)

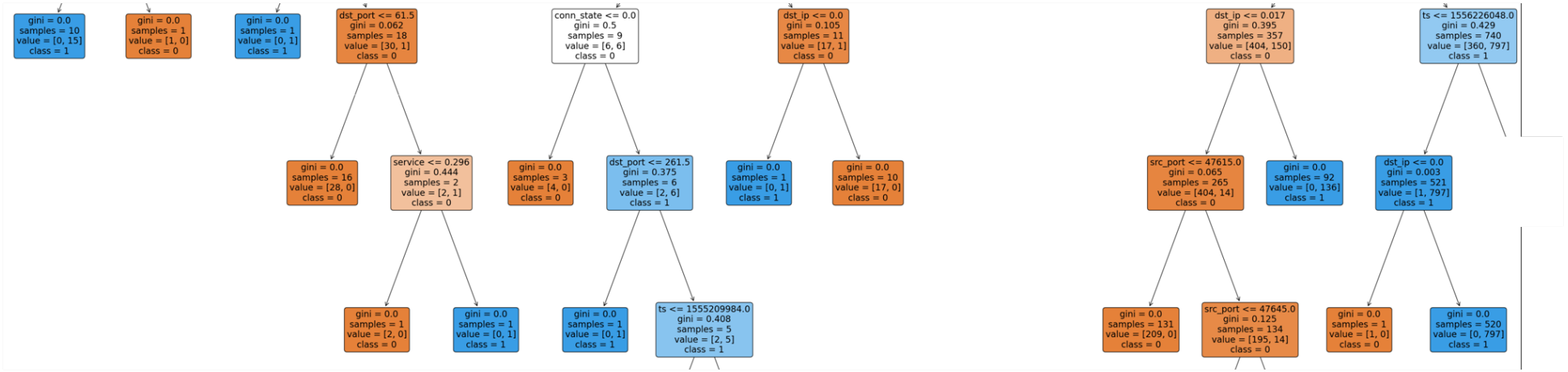


Figure A.6: Random Forest built using Train_Test_Network: part (e)

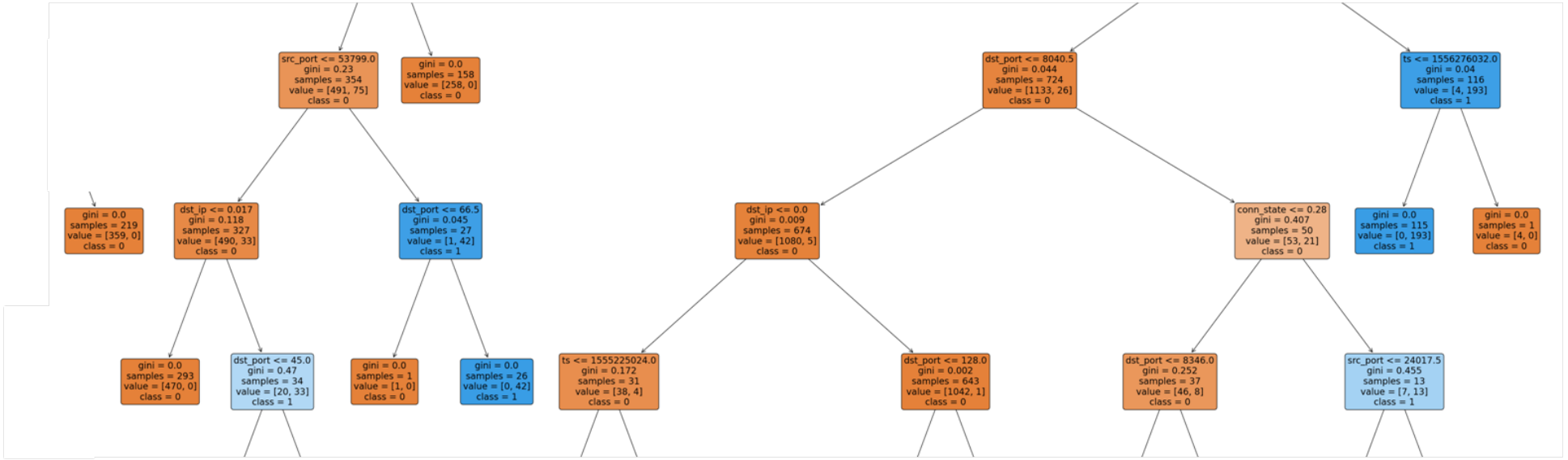


Figure A.7: Random Forest built using Train_Test_Network: part (f)

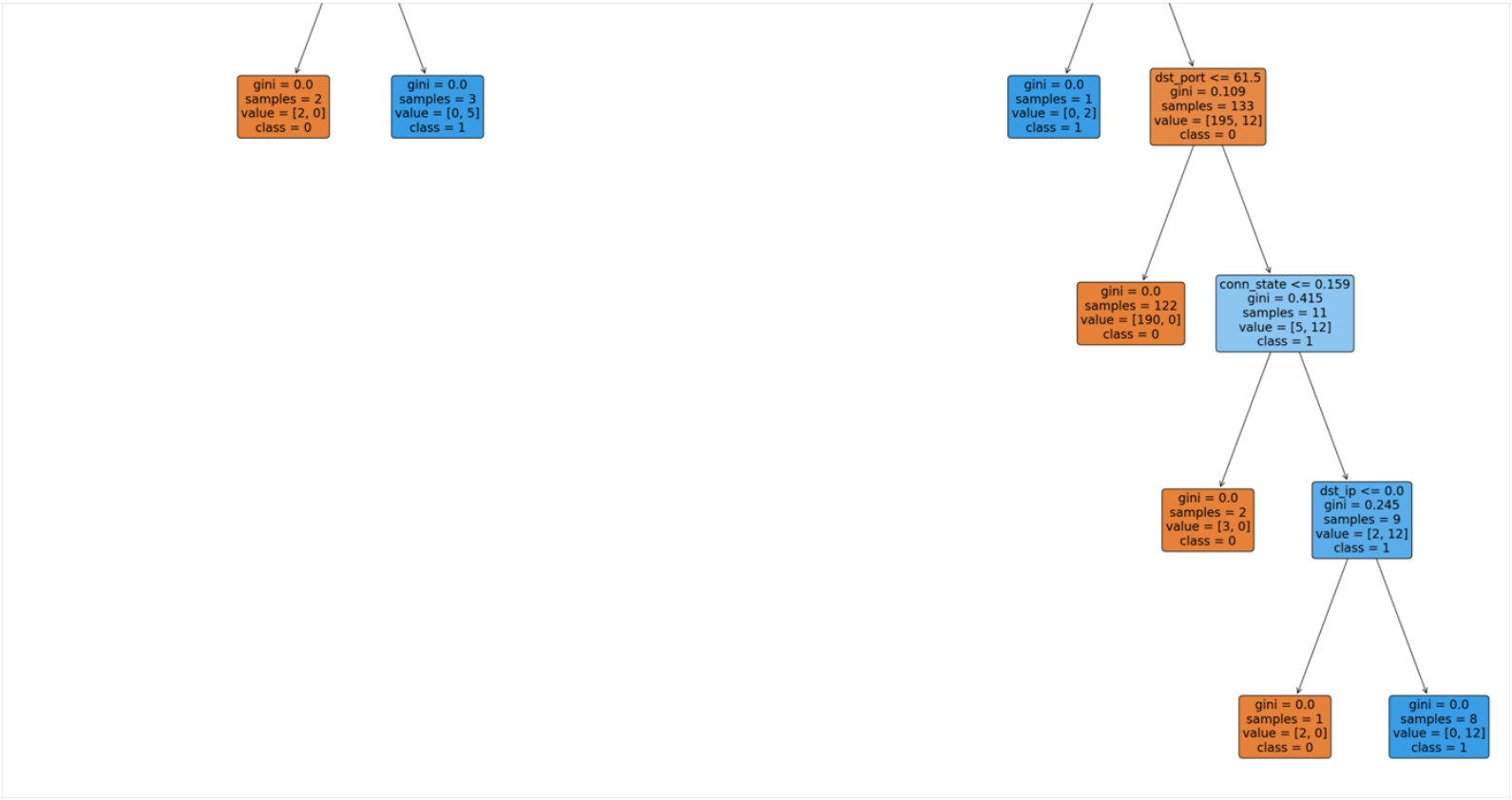


Figure A.8: Random Forest built using Train_Test_Network: part (g)

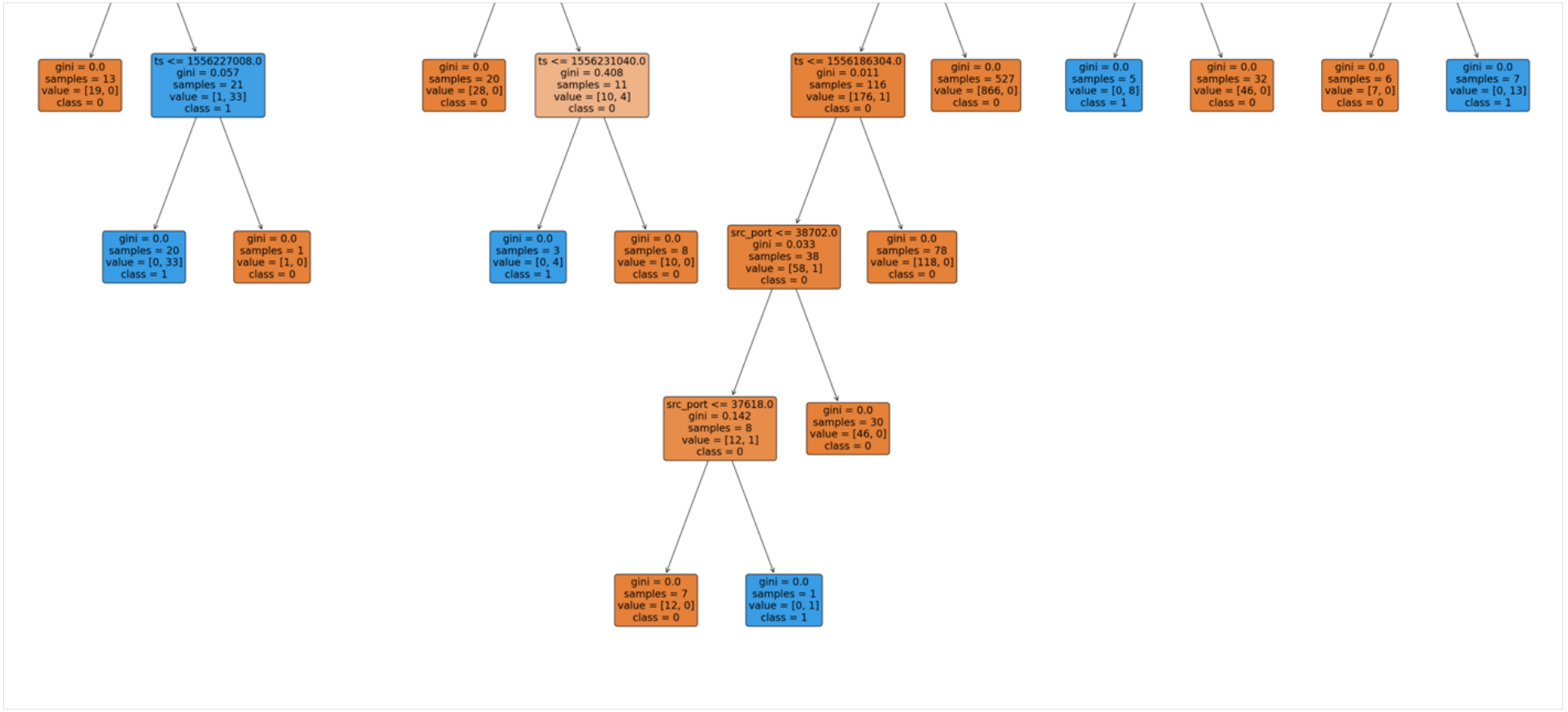


Figure A.9: Random Forest built using Train_Test_Network: part (h)

Appendix B
Random Forest Built using TON_IoT (Network_9) Dataset

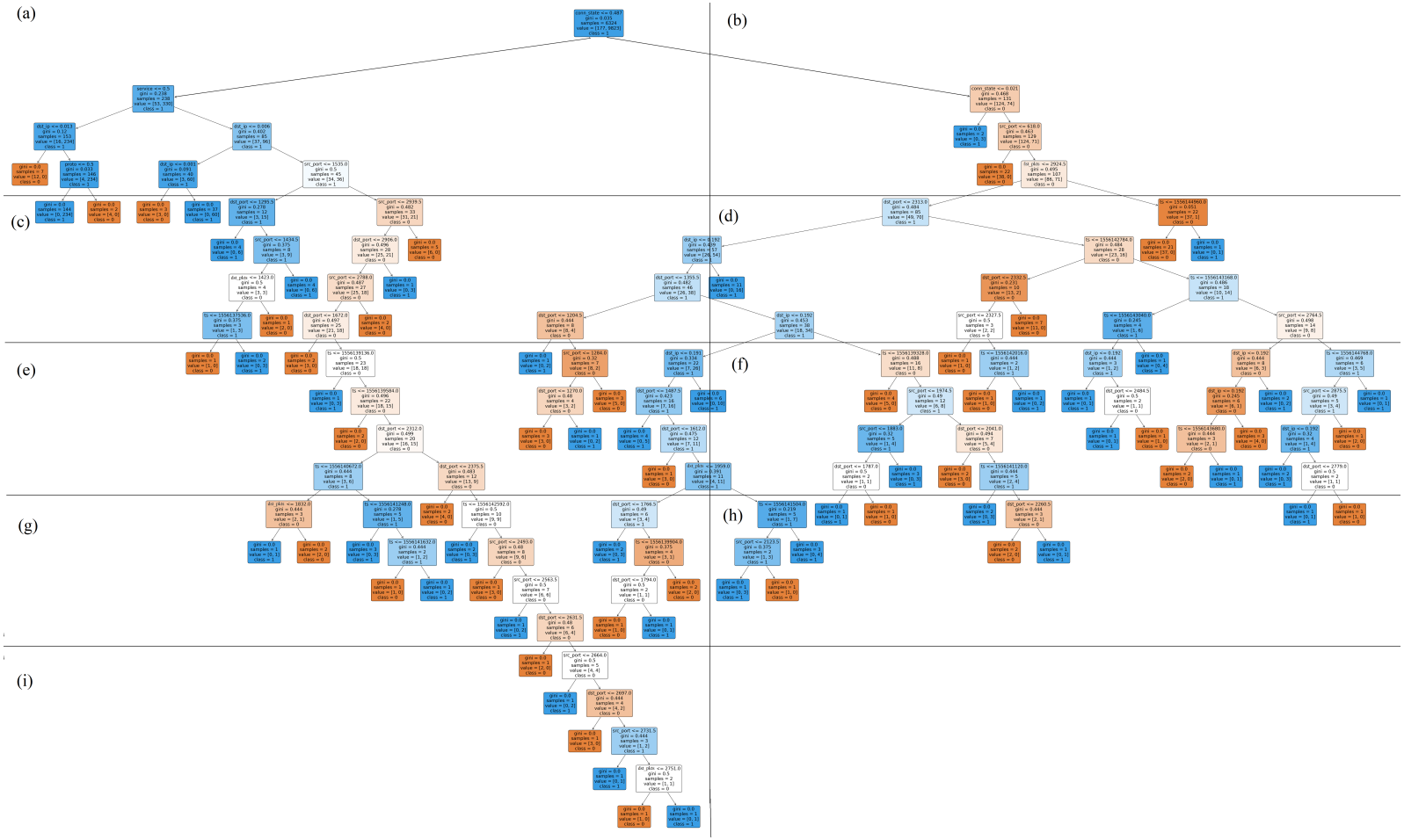


Figure B.1: Random Forest built using Network_9 file

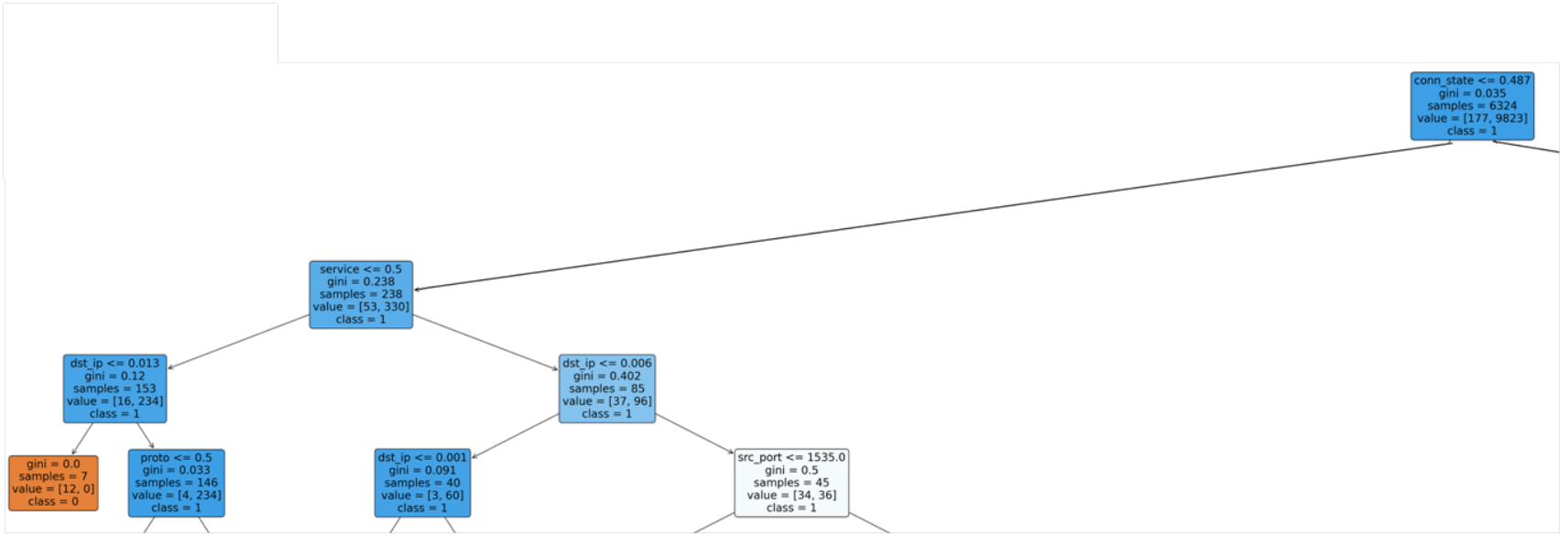


Figure B.2: Random Forest built using Network_9: part (a)

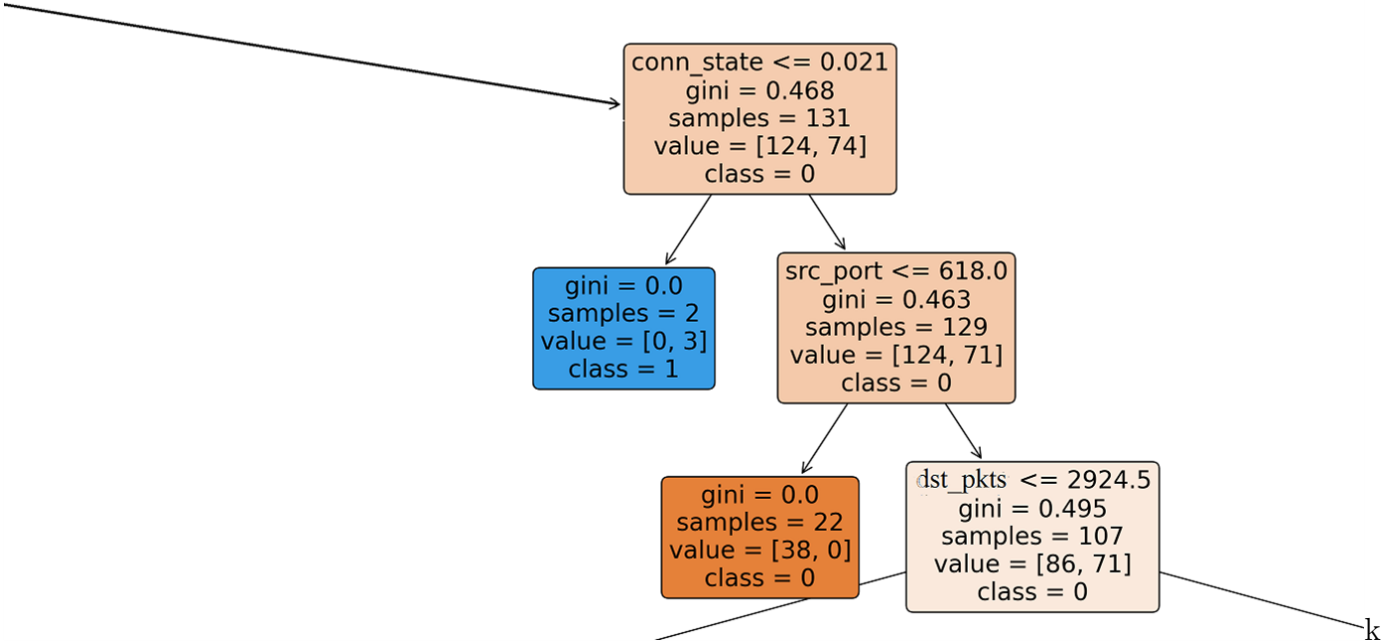


Figure B.3: Random Forest built using Network_9: part (b)

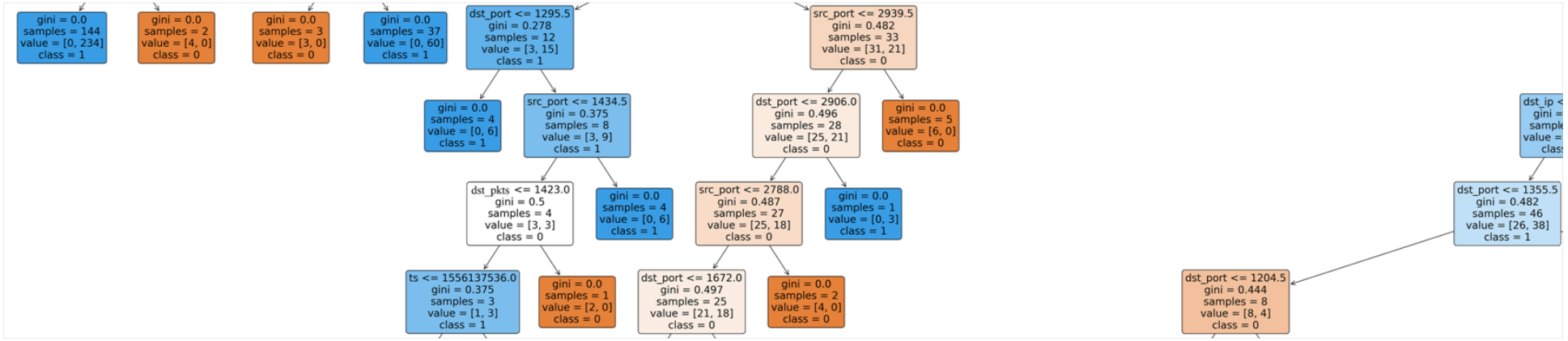


Figure B.4: Random Forest built using Network_9: part (c)

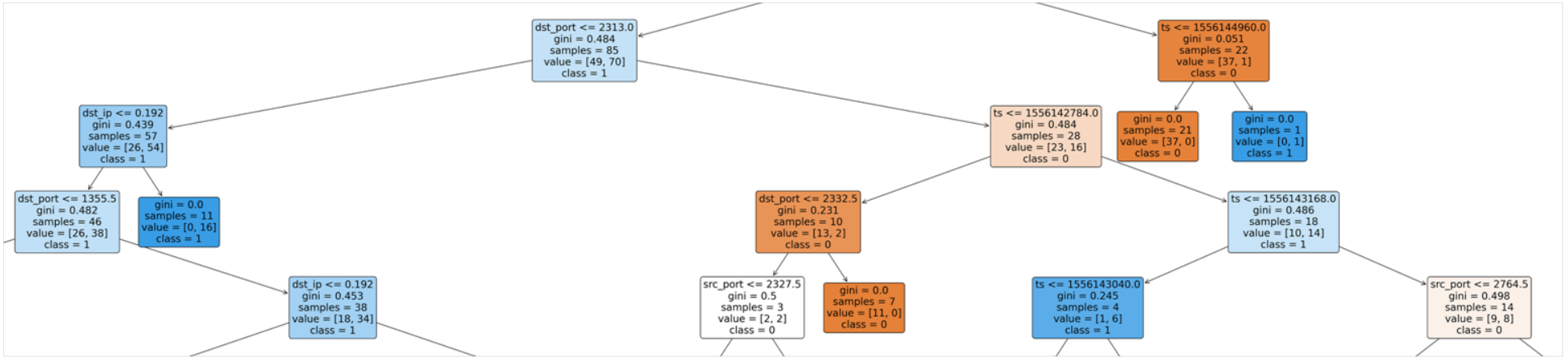


Figure B.5: Random Forest built using Network_9: part (d)

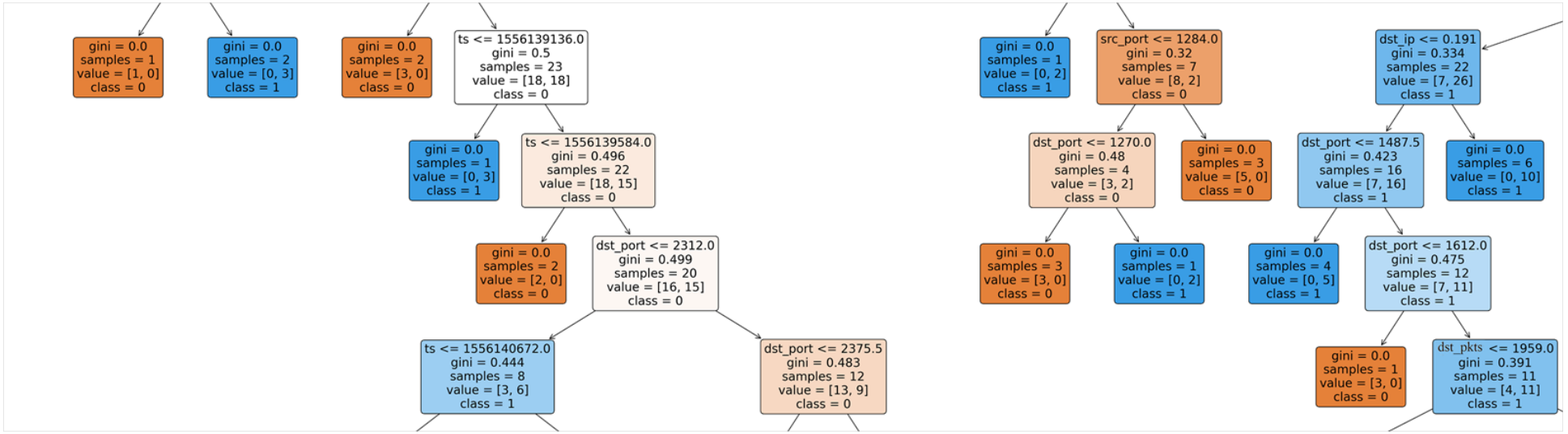


Figure B.6: Random Forest built using Network_9: part (e)

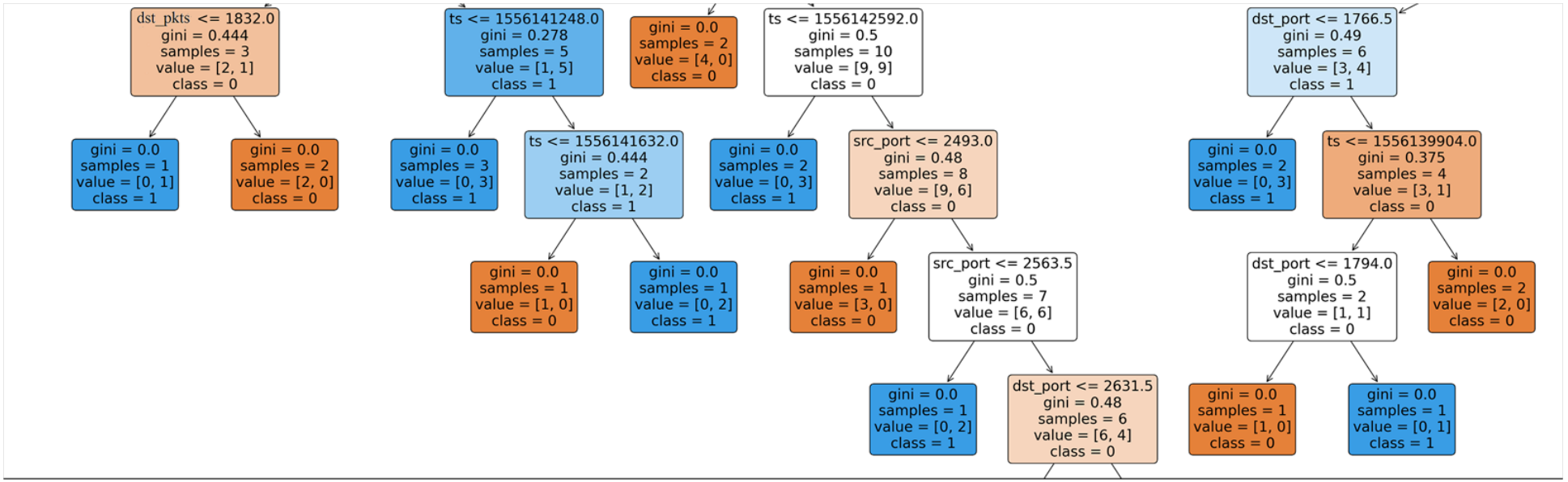


Figure B.7: Random Forest built using Network_9: part (f)

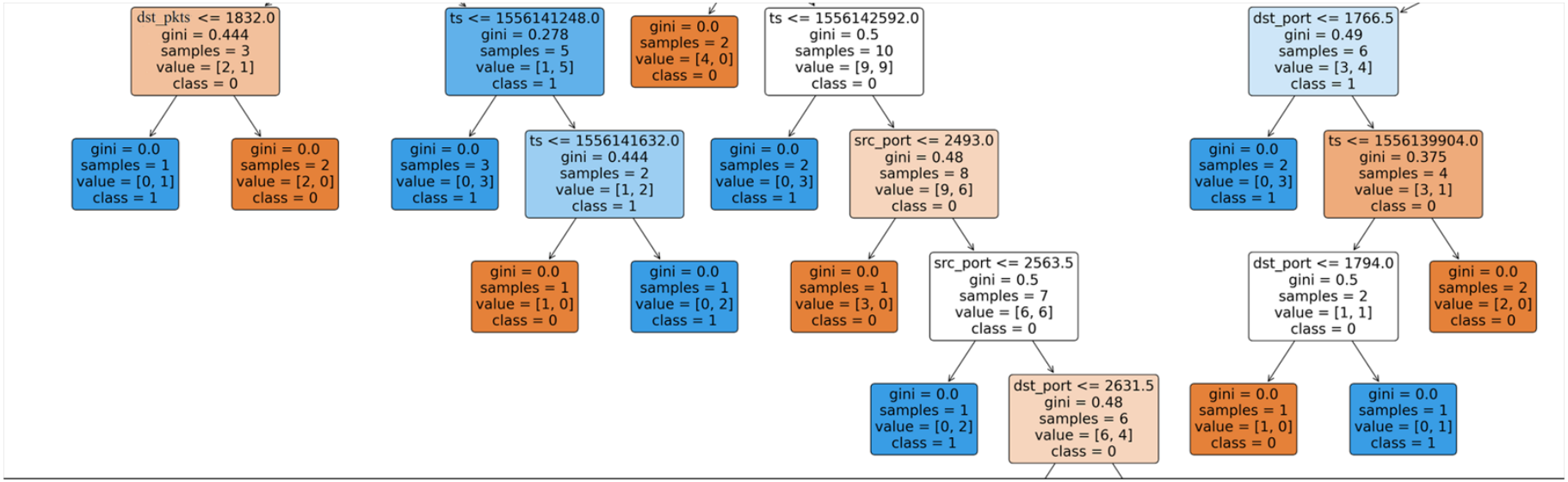


Figure B.8: Random Forest built using Network_9: part (g)

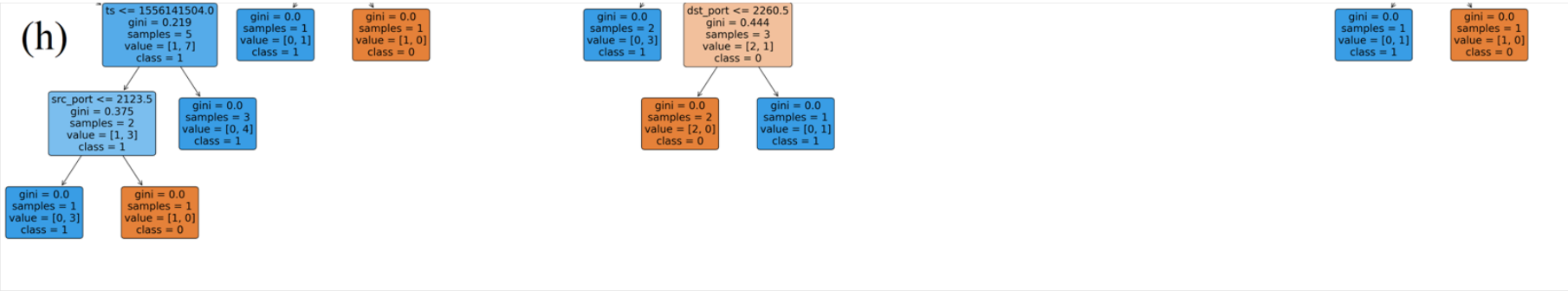


Figure B.9: Random Forest built using Network_9: part (h)

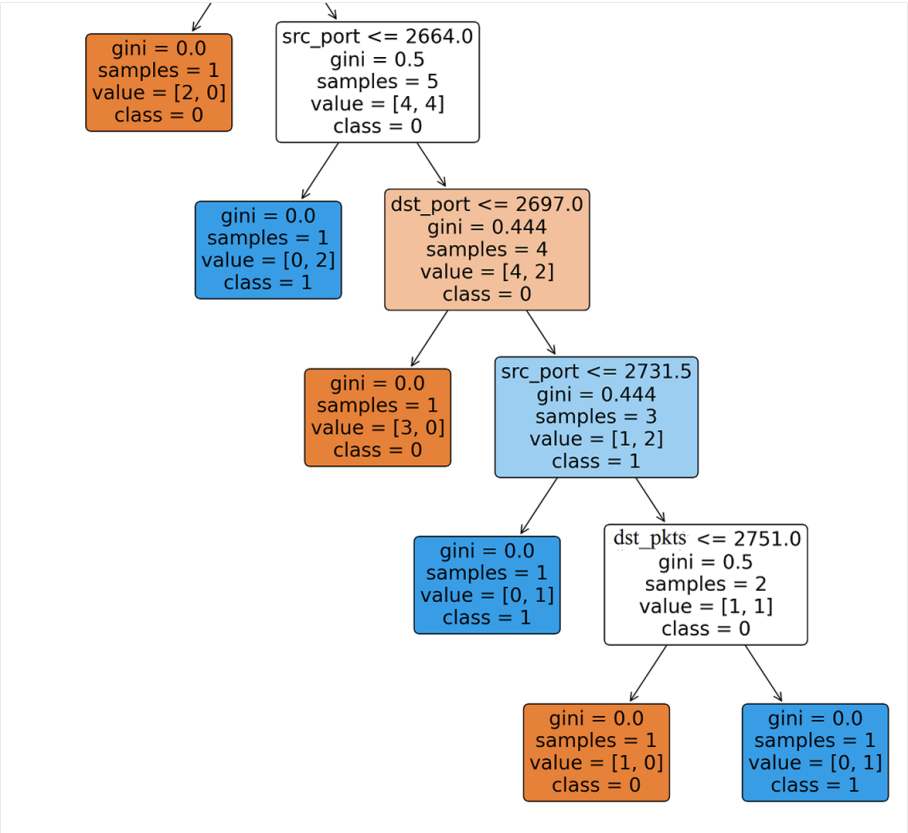


Figure B.10: Random Forest built using Network_9: part (i)