A Hybrid Machine Learning Intrusion Detection System Framework with Integrated Server and Client Models for Wireless Sensor Networks

by

Hongwei Zhang

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
April 2024

Table of Contents

List of Tables

List of Figures

Abstract

Federated Learning (FL) has emerged as a novel distributed Machine Learning (ML) approach to tackle the challenges associated with data privacy and overload in ML-based intrusion detection systems (IDSs). Drawing inspiration from the FL architecture, this thesis introduces the Server-Client Machine Learning Intrusion Detection System (SC-MLIDS), a hybrid ML IDS framework tailored for Wireless Sensor Networks (WSNs). SC-MLIDS is crafted to leverage ML for achieving a two-layer intrusion detection mechanism in WSNs, free from constraints posed by specific attack types. The framework follows a server-client model compatible with the configuration of sensor nodes, sink nodes, and gateways in WSNs. In this setup, client models located at sink nodes undergo training using sensing data, while the server model at the gateway is trained using network traffic data. This two-layer training approach not only amplifies the efficiency of intrusion detection but also ensures comprehensive network coverage.

The principal innovation of SC-MLIDS is the development of two model aggregation prediction algorithms, implemented at the gateway level. The first algorithm assesses models based on their performance metrics and assigned weights. The second algorithm uses a majority voting technique, combining predictions from both client and server models to bolster accuracy. In the operational phase, sensor nodes transmit collected data to their respective sink node for initial validation by the client model. Once the data is validated and associated with network traffic information, it is forwarded to the gateway for further validation through the model aggregation prediction algorithms.

The results of our simulation experiments corroborate the effectiveness of the proposed SC-MLIDS framework. It generates precise aggregation predictions, leading to a substantial reduction in redundant data transmissions. Furthermore, the SC-MLIDS framework exhibits efficacy in detecting intrusions through a two-layer validation process.

List of Abbreviations Used

Acronyms

AES Advanced Encryption Standard

AUC Area Under the ROC Curve

CPU Central Processing Unit

DBSCAN Density-based Spatial Clustering of Applications with Noise

DDoS Distributed Denial of Service

DNN Deep Neural Network

DNS Domain Name System

DoS Denial of Service

DT Decision Tree

ELM Extreme Learning Machine

EL Ensemble Learning

FedAvg Federated Averaging

FL Federated Learning

HTTP Hypertext Transfer Protocol

IDS Intrusion Detection System

IoT Internet of Things

IPv4 Internet Protocol version 4

IPv6 Internet Protocol version 6

IP Internet Protocol

KNN K-Nearest Neighbor

M2M Machine-to-Machine

ML Machine Learning

NB Naive Bayes

NFC Near Field Communication

PAN Personal Area Network

RBM Restricted Boltzmann Machine

RFID Radio Frequency Identification

RF Random Forest

ROC Receiver Operating Characteristic

SC-MLIDS Server-Client Machine Learning Intrusion Detection System

SSL Secure Sockets Layer

SVM Support Vector Machine

WSN Wireless Sensor Network

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Prof. Srinivas Sampalli, for his invaluable guidance and support during my study of a Bachelor's and Master's degree at Dalhousie University. He not only laid the foundation for my research and academic success with his extensive knowledge and insightful guidance, but his endless enthusiasm and optimism have also deeply influenced me to stay positive and motivated in the face of academic and life challenges. I look forward to continuing to learn from him and gain inspiration in my future academic journey.

I would like to express my deepest gratitude to my parents. Their unconditional love and support have been a solid backbone of my life. Their unlimited encouragement and trust have given me the courage and strength to face all challenges. With their unwavering support, my parents have enabled me to pursue my academic studies and achieve my dreams. Their understanding, patience, and generosity have provided me with the necessary resources and environment to realize my life goals. They are not only the most valuable treasure in my life but also a guiding light on my way to growth

I would like to extend my sincere thanks to Dr. Marzia Zaman from Cistel Technology. Throughout both my undergraduate and graduate studies, she has provided me with invaluable professional insights. Her challenging questions and critical guidance have significantly contributed to my research process, offering me essential support and advice. I have learned a great deal from her expertise and am deeply grateful for her assistance.

I would like to thank Dr. Darshana Upadhyay for her enthusiastic guidance on my research and her help in my teaching assistant work. I would also like to thank my classmates Richard Purcell and Nathanael Bowley in MYTech Lab for their inspiration, experience, and help during the early exploration and trial of this research.

Chapter 1

Introduction

The continuous advancement of Wireless Sensor Networks (WSNs) and their associated technologies has led to their widespread application in various industrial sectors. This evolution has significantly enhanced productivity through intelligent monitoring and management [1, 2]. However, WSNs face challenges in countering network attacks due to their inherent limitations. Researchers have proposed various intrusion detection systems (IDSs), focusing on identifying suspicious activities through strategies such as anomaly detection and misuse detection [3].

The emergence of Machine Learning (ML) has offered more possibilities for IDS in WSNs. Leveraging the inherent strengths of ML, these approaches have demonstrated enhanced performance in detecting network attacks, thus becoming a prominent focus in network security research. Through its capacity to train models with extensive relevant data, ML facilitates efficient, highly accurate, and automated risk assessment and intrusion detection [4].

Despite these advancements, current research in ML-based IDSs for WSNs primarily concentrates on employing specific ML algorithms to detect particular types of network attacks. This approach often results in solutions that are somewhat restrictive in their applications. Given the dynamic and diverse network environments that WSNs encounter in real-world scenarios, there is a critical requirement for designing a flexible, adaptable, and comprehensive ML IDS framework for WSNs.

In this thesis, we propose the Server-Client Machine Learning Intrusion Detection System (SC-MLIDS), a hybrid ML IDS framework for WSNs. It integrates server and client components, corresponding to the gateway and sink nodes in WSNs, and realizes two-layer validation of both sensing data and network traffic data. Inspired by the Federated Learning (FL) architecture, the SC-MLIDS framework is designed to be an adaptable solution that transcends limitations related to attack types, WSN architectural designs, ML algorithms, and model quantities. This two-layer approach

uses the inherent strengths of ML, achieving comprehensive intrusion detection that is not only efficient and highly accurate but also optimizes resource utilization.

The contributions of this thesis are summarized as follows:

1. **SC-MLIDS:** We have proposed a hybrid ML IDS framework, specifically designed based on the unique architectural characteristics and operational tasks of WSNs. This framework is not restricted to detecting specific types of attacks, thereby offering extensive intrusion detection capabilities for WSNs. By employing a simplified model at the sink node level, this framework enables the initial validation of sensing data. Furthermore, at the gateway, we employ aggregation prediction algorithms that merge the predictions of multiple models. This approach enables comprehensive validation of both sensing and network traffic data.

2. **Model Aggregation Prediction Algorithms:** We have proposed two distinct aggregation prediction algorithms aiming to merge prediction results by aggregating ML models generated from varied ML algorithms and tasks.

   (a) **Weighted Score Algorithm:** This algorithm combines the prediction results of each model based on their performance metrics and assigned weights. The resulting scores are then converted into binary classification results, ensuring a holistic and comprehensive prediction result.

   (b) **Majority Voting Algorithm:** This algorithm differentiates between primary and secondary models based on their task type and importance. It employs a voting mechanism among the classification results of the secondary model group, which generates the final prediction results when combined with the primary model's results.

To support open scientific inquiry and to facilitate replication and further research endeavours, the entire source code of the *SC-MLIDS*[1] project has been made publicly accessible. Interested readers and researchers are encouraged to visit the GitHub code repository. This repository contains all relevant code, algorithms, and datasets used in the development and evaluation phases of the SC-MLIDS project.

[1] https://github.com/Hongwei-Z/SC-MLIDS

The remainder of this thesis is structured as follows: Chapter 2 provides a comprehensive background and reviews related works. In Chapter 3, we detail the design of the proposed SC-MLIDS framework and its aggregation prediction algorithms, describing the dataset and its processing methods. Chapter 4 presents the experimental setup and the results obtained from our simulation. Chapter 5 discusses these results, exploring their implications and highlighting the limitations of this thesis while suggesting directions for future work. This thesis concludes with Chapter 6, which summarizes our findings and underscores the key contributions of our research.

Chapter 2

Background and Related Work

2.1 Wireless Sensor Networks (WSNs)

2.1.1 Overview

Wireless Sensor Networks (WSNs) are self-configuring networks that contain many wireless sensor nodes [5, 6]. These nodes are used for sensing and monitoring the environment in which they are deployed, and for collecting data. The WSN architecture, as shown in Figure 2.1, consists of the following components: sensor nodes, sink nodes and the gateway [7].

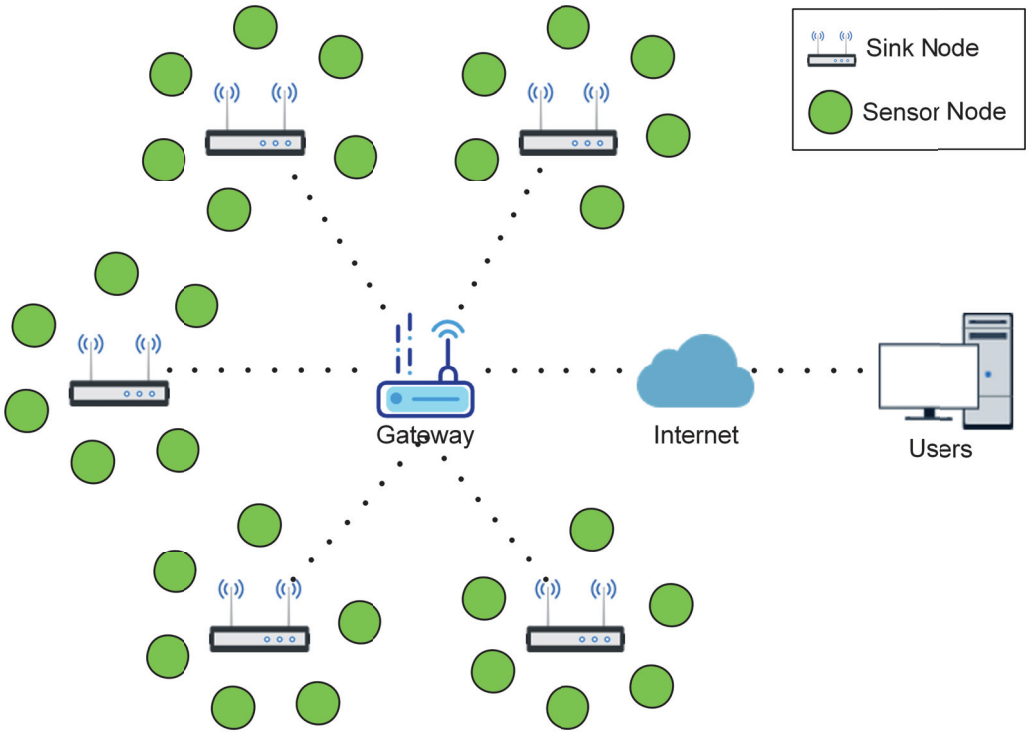

Figure 2.1: WSNs Architecture

Sensor nodes are tiny devices with low power consumption, integrated with sensors, microcontroller, transceiver, power supply, and operating system [8, 9]. The

sensor node senses and collects information from the environment through various types of equipped sensors. Depending on the application scenario, the required sensor types and data types vary. Common sensing data include temperature, humidity, pressure, light, and sound [10].

The microcontroller, acting as the Central Processing Unit (CPU) of the sensor node, controls all hardware components by running a micro-operating system [11, 12]. It performs the collection and simple processing of sensing data, then sends this data to the sink node via the transceiver, or communicates with other sensor nodes [8].

The sink node is the data collection unit [13], responsible for receiving and organizing the data sent from the wireless sensor nodes in the covered area. In WSN architectures, sink nodes often communicate with the gateway or the Internet to upload data received from sensor nodes [13, 14]. However, in certain WSN configurations, the gateway may be replaced by the sink node, allowing the sensor nodes to upload data directly via the sink node [15]. Such arrangements enhance the efficiency of data transmission. In more extensive WSN architectures, which include a large number of sensor nodes, the deployment of multiple sink nodes can further enhance data collection efficiency [14, 16]. By assigning each sink node responsibility for only a limited number of sensor nodes to communicate and process data, it is possible to avoid issues such as network latency and throughput limitations that arise when a single sink node is overwhelmed by communications from numerous sensor nodes [16].

The gateway receives the data generated by the sensor nodes, which is transmitted via the sink node, and uploads it to the Internet [17]. This process enables users to access, process, and analyze the data. The superior hardware configuration of the gateway facilitates the implementation of functions that are not feasible with sensor nodes due to various limitations. An example of such a function is security solutions to counter network attacks in WSNs [18].

Advances in technologies related to WSNs have led to their rapid application across various industries. In the military field, WSNs can provide services such as data collection, communication, and battlefield surveillance [19]. In the healthcare sector, WSNs enable continuous remote monitoring of the health statuses of patients without the constraints of fixed locations, resulting in improved quality of healthcare services [20]. WSNs facilitate intelligence in the agricultural field by monitoring

farms, monitoring temperature changes, and managing irrigation systems, thereby increasing agricultural productivity while reducing costs [21]. In the smart home sector, WSNs provide users with enhanced home security through monitoring of temperature, smoke, and gas leakages [22]. Additionally, WSNs offer cost-effective solutions to societal challenges, such as forest fire detection and air quality monitoring [23, 24].

WSNs are characterized by their small node size, low cost, and the capacity for widespread deployment. However, these features also result in several limitations, including computational power, power consumption, power supply, communication and sensing capabilities [25, 26]. Consequently, these limitations pose multiple challenges for WSNs, particularly in network security.

2.1.2 Attacks on WSNs

Common attacks on WSNs include Denial of Service (DoS) attacks, Sybil attacks, Blackhole / Sinkhole attacks, Hello Flood attacks, Wormhole attacks, and attacks compromising data integrity.

Denial of Service Attack

Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks are among the most common attacks on WSNs [27]. These attacks flood sensor nodes with excessive requests, resulting in them being unable to provide services properly [28]. Since the sensor nodes are restricted in terms of computational power and power supply, they are particularly vulnerable to DoS attacks that drain resources and power, thereby disabling the sensor node [29].

Sybil Attack

The Sybil attack involves generating and propagating multiple false identities by capturing or inserting malicious nodes to impersonate multiple legal nodes [28]. This type of attack is detrimental as it disrupts the routing protocols of WSNs [30]. It interferes with operations such as voting, reputation evaluation, and data aggregation. Consequently, it reduces data integrity, security, and resource utilization in WSNs [30, 31].

Blackhole / Sinkhole Attack

In the Blackhole / Sinkhole attack, a compromised malicious node intercepts the route requests from neighbouring nodes and responds with false shortest route paths leading to the sink node [32, 33]. As a result, these neighbouring nodes, misled by the false routing information, direct their packets to the malicious node [32]. This node, executing the Blackhole attack, either blocks or drops these packets. Such an attack, characterized by attracting traffic without forwarding it, can significantly impact the network latency and throughput of the WSNs [34].

Wormhole Attack

In the Wormhole attack, the attacker compromises two nodes within the WSN, establishing a direct high-speed communication tunnel between them [35]. The malicious node advertises to nearby nodes that it possesses the shortest route, thereby inducing these nodes to transmit their packets through the tunnel created by the malicious node [36]. This transmission leads the nodes on both ends of the tunnel to mistakenly believe they are neighbouring nodes [37]. As a result, the malicious node attracts nearby traffic, thereby adversely affecting the routing algorithm and confusing the routing protocol [37, 38].

Hello Flood Attack

The Hello Flood attack exploits a vulnerability in the WSN initialization mechanism, designed to identify neighbouring nodes through exchanging greeting messages [39]. Typically, a node sends a "Hello" message to its neighbouring nodes, enabling the message receiver to recognize the sender as a neighbouring node [40, 41]. In this attack, the attacker employs a laptop-class device with high-power broadcast routing capabilities to send a message to the nodes in the WSN [39]. Consequently, these nodes mistakenly recognize the attacker as a neighbouring node. When a node transmits packets to the base station, they are routed through the attacker, creating a unidirectional link between the node and the attacker [41]. This situation leads to confusion in the network and disrupts its normal functioning [39].

Data-oriented Attack

In WSNs, the processes of data generation and packet transmission are susceptible to common wireless network attacks. Compromised sensor nodes may generate false data or selectively send packets, thereby compromising the integrity and reliability of the data [28]. Furthermore, during data transmission, the network traffic can be monitored, leading to potential data leakage [35]. Additionally, the transmitted data is at risk of being intercepted, altered, or discarded, which further undermines the reliability of the received data.

2.2 Internet of Things (IoT)

Internet of Things (IoT) forms a network infrastructure with many sensing, communication, network, and information processing devices [42, 43]. This infrastructure facilitates data exchange among devices, between devices and servers, and between devices and the Internet [44].

The fundamental technology of IoT is Radio Frequency Identification (RFID), which allows devices embedded with RFID tags to be read through Near Field Communication (NFC) technology [43, 45]. This capability enables unique identification, tracking, and monitoring of the devices [42]. IoT has achieved significant advancement with the introduction of additional technologies such as WSNs, Machine-to-Machine (M2M) communication, and low-power Personal Area Networks (PANs), thereby becoming more intelligent and interconnected [46].

IoT has found extensive applications across various sectors of human society, including intelligent control and monitoring in industry and agriculture, smart cities, smart homes, and healthcare [47, 48, 49].

2.3 Federated Learning (FL)

Federated Learning (FL) is a distributed Machine Learning (ML) approach that involves model training through the collaboration of multiple clients with a server [50, 51].

The FL architecture is shown in Figure 2.2 and the process can be summarized in six steps [50, 51, 52]:

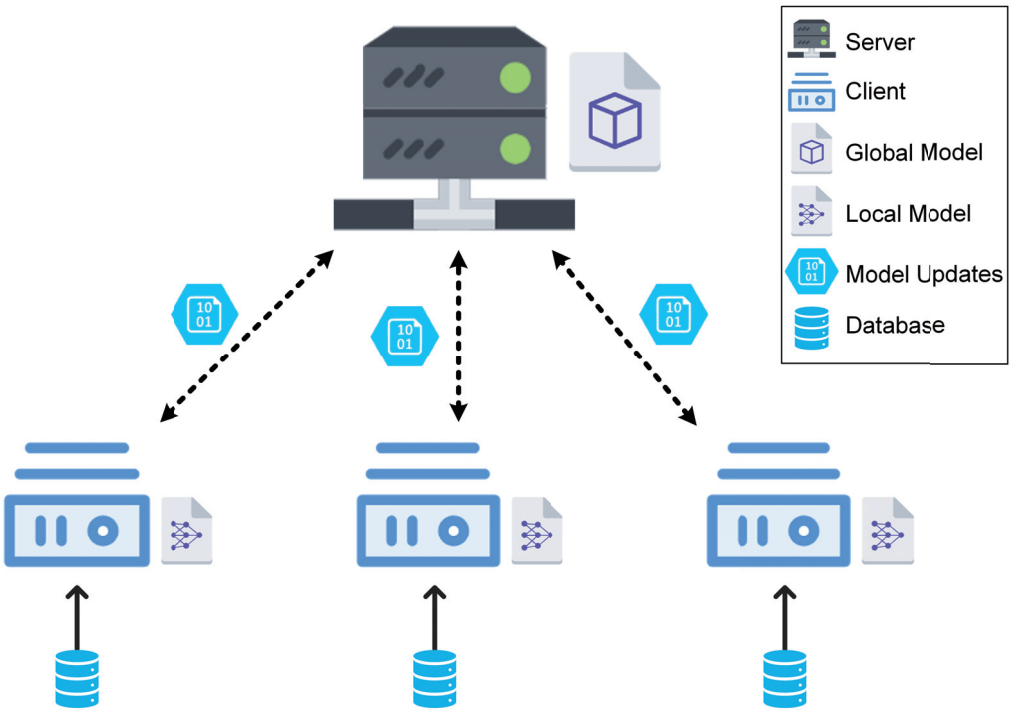

Figure 2.2: Federated Learning Architecture

1. A generic initialization model is configured as the global model on the server.
2. This global model is then distributed to all clients.
3. Clients use their local data to train the model, thus generating customized local models.
4. The parameters of the trained model, which differ from those of the global model, are sent from the clients to the server.
5. Upon receiving these parameters, the server aggregates them using a strategy such as Federated Averaging (FedAvg), allowing for the updating of the global model.
6. Repeat steps 4 and 5, with the updated global model's parameters sent back to the clients for subsequent rounds of training. The cycle continues until the model converges.

FL is well-suited for IoT applications, due to its decentralized nature, data privacy protection, and resource conservation [53, 54, 55]. The rapid expansion of IoT has

led to an enormous volume of data generated by an enormous of IoT devices, presenting challenges related to data overload, data security, and data privacy [53]. The distributed feature of FL eliminates the need for centralized data processing. Configuring numerous local clients not only alleviates computational burdens but also avoids privacy and security issues associated with data transmission. Therefore, FL offers an effective and privacy-preserving solution for IoT environments.

2.4 Ensemble Learning (EL)

Ensemble Learning (EL) represents a predictive methodology that combines multiple ML models, as illustrated in Figure 2.3. This approach involves training classifiers on the same dataset using different algorithms, thereby generating different models [56]. The primary objective is to synthesize a new model that integrates the strengths of distinct classification algorithms. This integration aims to transcend the limitations inherent in relying on a single classification algorithm, thereby enhancing the performance of predictions [57].

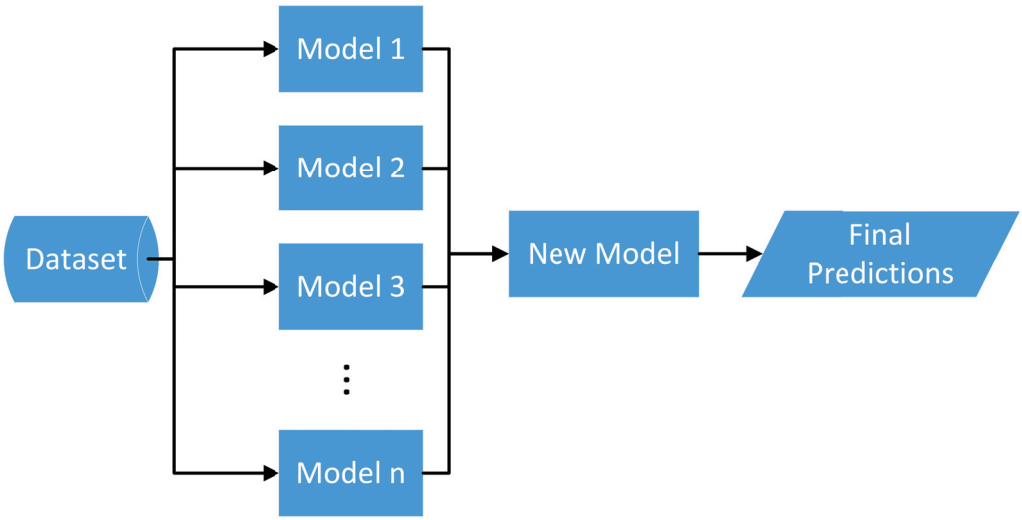

Figure 2.3: Ensemble Learning Architecture

EL can be categorized into three principal types: bagging, stacking, and boosting [58]:

1. **Bagging:** Bagging uses a randomized sampling technique to extract a subset of the dataset for model training [57]. The prediction results of individual

models are combined to generate the final predictions. This combination uses aggregation methods such as classification voting or regression averaging [59]. Within this framework, the majority voting method consolidates the prediction results from each model, and the class receiving the most votes is designated as the final predictions [57, 60]. An extension of this method, weighted majority voting, introduces model weights based on performance, assigning greater predictive priority to models demonstrating superior performance [57, 60]. The prediction results are then normalized to convert voting results into classification results [57].

2. **Stacking:** In this approach, the initial dataset is used to train cardinal learners. The outcomes generated by these cardinal learners are then used to train a meta learner [56, 57].

3. **Boosting:** This method involves iteratively training the learner with samples incorrectly predicted in previous iterations. A weak learner is progressively transformed into a strong learner through this process [56, 57].

2.5 Related Work

Traditional intrusion detection systems (IDS) for WSNs are classified into anomaly detection and misuse detection [3]. Anomaly detection systems primarily focus on identifying anomalies in nodes, networks, and data within WSNs [61]. Misuse detection involves the detection of known attacks through the pre-configuration of attack signatures [3]. However, the evolution of ML techniques has introduced more possibilities for IDS in WSNs. ML-based IDS for WSNs can effectively identify and filter anomalous data, thereby saving computational and network resources. Furthermore, such systems can autonomously detect, learn from, and prevent various network attacks and vulnerabilities, eliminating the need for human intervention [62].

Current research in ML-based IDS for WSNs primarily concentrates on deploying specific ML algorithms for the identification and detection of particular types of network attacks. For example, studies [62, 63] introduce methods using ML algorithms such as the K-Nearest Neighbors (KNN), Decision Trees (DT), and Support Vector

Machines (SVM), for outlier detection and the identification of specific network attacks. Concurrently, the advancement of ML has promoted innovative approaches such as Boosting and Deep Learning (DL) based schemes. For instance, research in [64] investigates the comparison of three novel Boosting methods against three traditional ML techniques for detecting network attacks in WSNs. Additionally, [65] explores the use of Deep Neural Networks (DNN) to solve the limitations inherent in traditional ML, especially in response to imbalanced attacks.

Given the potential constraints associated with single ML classifiers, some studies have advocated for model ensemble strategies to enhance detection capabilities. For example, [66] introduces an IDS scheme that integrates Random Forest (RF), Density-based Spatial Clustering of Applications with Noise (DBSCAN), and Restricted Boltzmann Machine (RBM). To enhance detection efficiency, certain methods, such as the one proposed in [67], apply feature selection algorithms to filter key features. This approach is aimed at reducing the time required for attack detection and ensuring efficient identification of attacks. Additionally, research efforts have been directed towards developing detection methods for specific data types. The study in [68] focuses on detecting anomalies in sensing data, while [69] combines traditional ML and DL techniques to identify malicious nodes through network traffic classification.

The aforementioned studies represent the main research direction of ML-based intrusion detection solutions for WSNs, offering valuable insights into WSN security. However, these researches mainly focus on applying specific models or algorithms to address unique network security challenges. There exists a notable gap in their consideration of comprehensive network security issues in real-world WSN application scenarios. The following key papers have made contributions to this research area and provided crucial foundations and inspirations for our SC-MLIDS.

Yu and Tsai [70] propose an innovative ML-based IDS for WSNs that transcends the limitations of detecting specific types of attacks. In their proposed framework, each sensor node is equipped with an Intrusion Detection Agent (IDA). These IDAs function autonomously, facilitating intrusion detection through two primary internal components. The first, the Local Intrusion Detection Component (LIDC), is responsible for analyzing local features to evaluate whether the host node is under

attack. The second, the Packet-based Intrusion Detection Component (PIDC), focuses on identifying malicious nodes by monitoring the communication activities of neighbouring nodes deemed suspicious by the LIDC. Alerts identified by the IDAs are transmitted to and examined by users through the base station. Importantly, false alarms detected prompt the base station to fine-tune and optimize the model, thereby enhancing its accuracy and reliability. This automation feature within the IDS significantly enhances its adaptability.

Medhat et al. [71] develop two algorithms for defining detection rules, both based on DT. These algorithms were specifically proposed for two WSN configurations designed by the researchers: one is base station-sink-sensor architecture, and the other is sink-sensor architecture. Within these configurations, the IDS is implemented at the sensor nodes level using either supervised or unsupervised learning algorithms. These algorithms are employed to generate detection rules through feature selection, corresponding to each of the two WSN configurations. Intrusion detection is then achieved by applying DT classifiers, which use the defined rules to detect anomalies in sensing data. The binary DT-based algorithms are notable for their high detection accuracy, achieved with a reduced number of features and lower resource demands.

Zhang et al. [72] introduce a hierarchical IDS based on the Extreme Learning Machine (ELM) methodology. In their proposed framework, the cluster head node initially preprocesses data collected by sensor nodes. This data is then transmitted to the sink node for feature extraction relevant to intrusion detection, and subsequently forwarded to the management node over the Internet. Within the management node, the intrusion detection module employs their proposed Multi-Kernel ELM (MK-ELM) model to determine the presence of intrusions. The results generated from this model are then forwarded to an anomaly processing module for further analysis and decision-making. Simulation results of their model have demonstrated its capability to achieve high-precision detection while concurrently reducing the time required for detection.

Maleh et al. [73] propose an IDS that mixes SVM-based anomaly detection and predefined attack rules signature, thereby enhancing the system's capability to identify attacks or anomalies. This system is designed to integrate into a clustered WSN topology, featuring dynamically elected cluster heads responsible for monitoring and authenticating WSN nodes. This hybrid approach effectively combines the strengths

of both ML anomaly detection and signature rules-based IDSs, resulting in a solution that is not only lightweight and efficient but also highly accurate.

Alruhaily and Ibrahim [74] propose a two-layer ML defence strategy for IDS in WSNs. A Naive Bayes (NB) classifier is employed at the sensor nodes layer to make an initial classification decision on packets, labelling them as normal or malicious traffic. For in-depth analysis, malicious traffic is examined in the cloud using an RF multi-class classifier. This classifier is critical in identifying the specific attack type and guiding the implementation of corresponding defence mechanisms. Their approach achieves efficient utilization of network resources and demonstrates high accuracy in experiments designed to detect normal traffic and four distinct types of attacks.

A review of prior research has highlighted the significant improvements made in hybrid ML-based IDSs for WSNs. However, some critical shortcomings are still present. Current research has focused on the use of specific ML algorithms or the combination of ML algorithms with traditional IDS approaches (e.g., signature rules) to identify specific attack types. Additionally, these studies have notable limitations in terms of ML algorithm selection, model optimization, processing efficiency, resource consumption, and deployment adaptability in WSNs, especially in dynamically changing network environments and diverse security threats. These gaps highlight the importance of the need for a hybrid ML IDS framework that is not only highly flexible and adaptable but also capable of transcending the limitations of attack types.

To address these gaps, this thesis proposes the Server-Client Machine Learning Intrusion Detection System (SC-MLIDS) framework. The SC-MLIDS framework is designed based on the data characteristics and architectural features of WSNs. By implementing a two-layer detection mechanism, it aims to achieve lightweight intrusion detection that is both highly accurate and efficient. The adoption of a server-client architecture not only ensures that the framework is highly adaptable and scalable but also allows the framework to flexibly adapt to dynamically changing network security threats and requirements. In addition, the framework supports diverse algorithms and model adjustment, which can adapt to different model combinations generated by diverse ML algorithms to meet different WSN architectures and application requirements.

Chapter 3

Proposed Framework

3.1 Datasets

3.1.1 Dataset Preprocessing

Due to the unique characteristics of the framework proposed in this thesis, specific requirements are set for the dataset to be used. The dataset must be generated in a Wireless Sensor Network (WSN) scenario, including sensing and network traffic data, and labelled according to network attacks. However, finding relevant datasets for WSNs that meet these criteria is challenging. Since the WSN is one of the fundamental technologies supporting the Internet of Things (IoT), and both involve wireless devices equipped with sensors, selecting a suitable IoT dataset presents an optimal alternative.

The TON_IoT [75] dataset contains data from IoT and Industrial IoT sensor devices, as well as operating system and network traffic data, making it suitable for the validation of Machine Learning (ML) network security solutions for IoT, such as intrusion detection systems (IDSs). The TON_IoT dataset contains a large amount of network attack traffic data collected from large-scale real-world networks targeting the IoT and is therefore popular in IoT security research.

The TON_IoT dataset is divided into four categories: the raw dataset, the processed dataset, the dataset split according to training and testing sets, and the description files of the dataset. The raw dataset contains sensing data from many IoT sensors, captured network traffic data, and data from host run-time processes on Linux and Windows operating systems.

In this thesis, the IoT device sensing data and network traffic data from the processed dataset are employed. The weather dataset within the IoT device sensing data, which closely aligns with the WSN scenario, is the only dataset used. Conversely, the network traffic dataset is used in its entirety, as it is not categorized based on specific

IoT devices.

The IoT weather dataset, as presented in Table 3.1, collects weather sensing data including temperature, pressure, and humidity between March 31 and April 27, 2019, and is categorized by attack type. A label of '0' denotes normal traffic and a label of '1' indicates malicious traffic.

Table 3.1: IoT Weather Dataset

Date	**Time**	**Temperature**	**Pressure**	**Humidity**	**Label**	**Type**
31-Mar-19	12:36:52	31.788508	1.035	32.036579	0	normal
31-Mar-19	12:36:53	41.630997	1.035	30.886165	0	normal
31-Mar-19	12:36:54	42.256959	1.035	19.755908	0	normal
31-Mar-19	12:36:55	49.116581	1.035	78.949621	0	normal
31-Mar-19	12:36:56	24.017085	1.035	40.001059	0	normal
...	...	...	...	...	...	...
27-Apr-19	12:41:17	40.384291	9.049059	93.094490	0	normal
27-Apr-19	12:41:18	47.240113	-5.782022	28.146511	0	normal
27-Apr-19	12:41:18	23.540606	0.913648	30.478316	0	normal
27-Apr-19	12:41:18	46.016150	3.493588	73.328413	0	normal
27-Apr-19	12:41:20	48.082975	0.100271	20.949148	0	normal

The dataset has a total of 650,242 rows, and the distribution of their types can be seen in Table 3.2. Of these, 86%, amounting to 559,718 rows, represent normal data, while the remaining 14% represent data affected by each of the seven types of attacks.

Table 3.2: Attack Type Distribution of the IoT Weather Dataset

Attack Type	**Count**
normal	559718
backdoor	35641
password	25715
ddos	15182
injection	9726
ransomware	2865
xss	866
scanning	529

The network traffic dataset is notably complex, containing network traffic from various devices. It consists of 21,978,631 rows and 46 columns across 23 dataset files. The data is categorized into several types: connection activity, statistical activity,

Domain Name System (DNS) activity, Secure Sockets Layer (SSL) activity, Hypertext Transfer Protocol (HTTP) activity, violation activity, and data labels.

Table 3.3: Processed Network Dataset

index	0	1	2	3	...
ts	1554198358	1554198358	1554198359	1554198359	...
src_ip	3.122.49.24	192.168.1.79	192.168.1.152	192.168.1.152	...
src_port	1883	47260	1880	34296	...
dst_ip	192.168.1.152	192.168.1.255	192.168.1.152	192.168.1.152	...
dst_port	52976	15600	51782	10502	...
proto	tcp	udp	tcp	tcp	...
service	-	-	-	-	...
duration	80549.53026	0	0	0	...
src_bytes	1762852	0	0	0	...
dst_bytes	41933215	0	0	0	...
conn_state	OTH	S0	OTH	OTH	...
src_pkts	252181	1	0	0	...
src_ip_bytes	14911156	63	0	0	...
dst_pkts	2	0	0	0	...
dst_ip_bytes	236	0	0	0	...
label	0	0	0	0	...
type	normal	normal	normal	normal	...

For this thesis, connection activity, statistical activity, and data labels are of primary importance, therefore it is unnecessary to use the entire dataset. The refined data, as illustrated in Table 3.3, includes information such as timestamps, source and destination ports, protocols, duration, byte counts, connection states, packet lengths and counts, and data labels, with a total of 17 columns.

Table 3.4: Attack Type Distribution of the Processed Network Dataset

Attack Type	**Count**
scanning	7140161
ddos	6165008
dos	3375328
xss	2108944
password	1365958
normal	788599
backdoor	508116
injection	452659
ransomware	72805
mitm	1052

The distribution of these types is detailed in Table 3.4, where only about 3.6% of the data, corresponding to 788,599 rows, is normal, with the remainder of each of the nine types of attacks.

The network traffic dataset covers a broader scope than the IoT weather dataset, so they need to be merged based on shared features. First, the timestamps in the network traffic dataset are converted to match the date and time format of the IoT weather dataset. The time frame for the network traffic dataset spans from April 2 to April 29, 2019. Consequently, data before April 2 is removed from the IoT weather dataset, leaving approximately 53% of its data, equivalent to 347,223 rows.

The common features between both datasets include time-date, attack type, and data labels. To facilitate a more precise merger, the datasets are merged using a left merge method based on the IoT weather dataset, focusing on time-date and attack type as the merging features. The merged dataset contains 3,256,615 rows. Due to the non-uniqueness of the combination of time-date and attack type, a substantial volume of replicated data is generated and requires further cleaning.

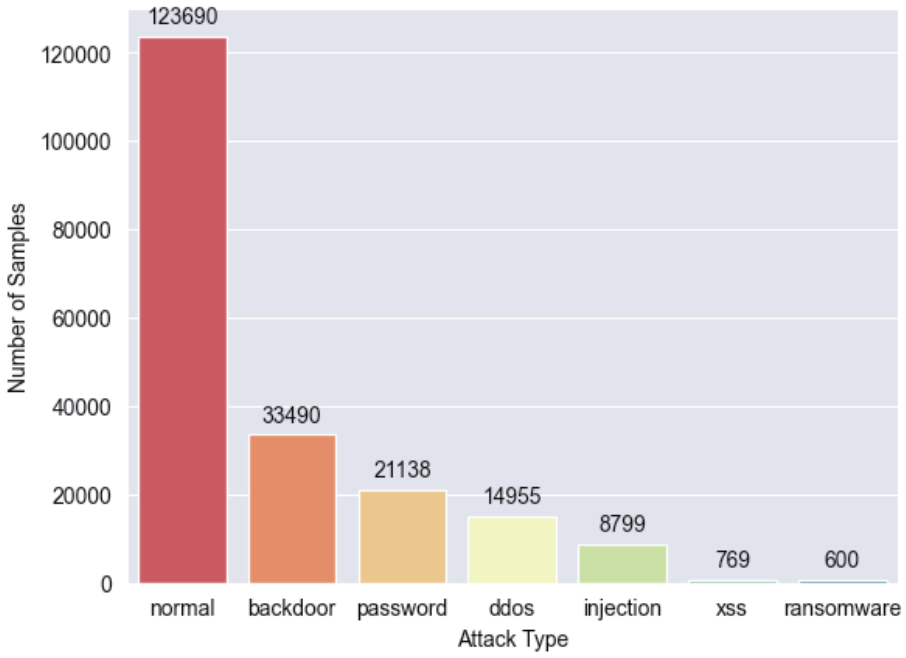

Figure 3.1: Attack Type Distribution of the Merged Dataset

The cleaning process involves two key steps. First, rows containing non-numeric elements (NaN), indicative of missing values from the dataset merging, are removed.

Second, within the merged dataset, rows that simultaneously match all five features (time-date, temperature, pressure, humidity, and attack type) of the baseline IoT weather dataset and are duplicates are also removed. Following these steps, the cleaned merged dataset contains 203,441 rows. Compared to the date-filtered IoT weather dataset, this represents 58.59% of the data.

Figure 3.1 illustrates the distribution of attack types in the cleaned merged dataset. Normal data constitutes approximately 60.80% of the dataset, amounting to 123,690 rows. Malicious traffic proportions include 16.46% for backdoor traffic, and 10.39% for password traffic. With the remaining four types of malicious traffic occupying smaller percentages.

The merged dataset requires further processing to prepare it for use in ML models. The IP addresses in the network traffic dataset, which included both IPv4 and IPv6 formats, could not be easily identified and encoded. Therefore, they are removed. The non-numeric data in the merged dataset is encoded using a label encoder, converting character data into numeric format. The numeric data is scaled using a scaler, compressing it into a range from 0 to 1. After shuffling the merged dataset randomly, the dataset preprocessing is completed.

Table 3.5: Merged Dataset

index	0	1	2	3	4	...	203440
temperature	0.9486	0.5170	0.9596	0.5819	0.5132	...	0.9509
pressure	0.6904	0.5001	0.4670	0.6743	0.7438	...	0.4022
humidity	0.2075	0.4147	0.4953	0.5789	0.7367	...	0.6414
src_port	0.8152	0.6586	0.8236	0.1202	0.8618	...	0.7081
dst_port	0.0068	0.0008	0.1603	0.6426	0.1233	...	0.0012
proto	1	2	1	1	1	...	1
service	9	3	0	0	0	...	0
duration	0	0	0	0	0	...	0
src_bytes	0	0	0	0	0	...	0
dst_bytes	0	0	0	0	0	...	0
conn_state	4	10	0	0	1	...	10
src_pkts	0	0	0	0	0	...	0
src_ip_bytes	0	0	0	0	0	...	0
dst_pkts	0.0001	0	0	0	0	...	0
dst_ip_bytes	0.0001	0	0	0	0	...	0
target	1	0	0	0	1	...	1

The resulting dataset is presented in Table 3.5. It combines sensing data from the

IoT weather dataset with network connection activity and statistical activity data from the network traffic dataset.

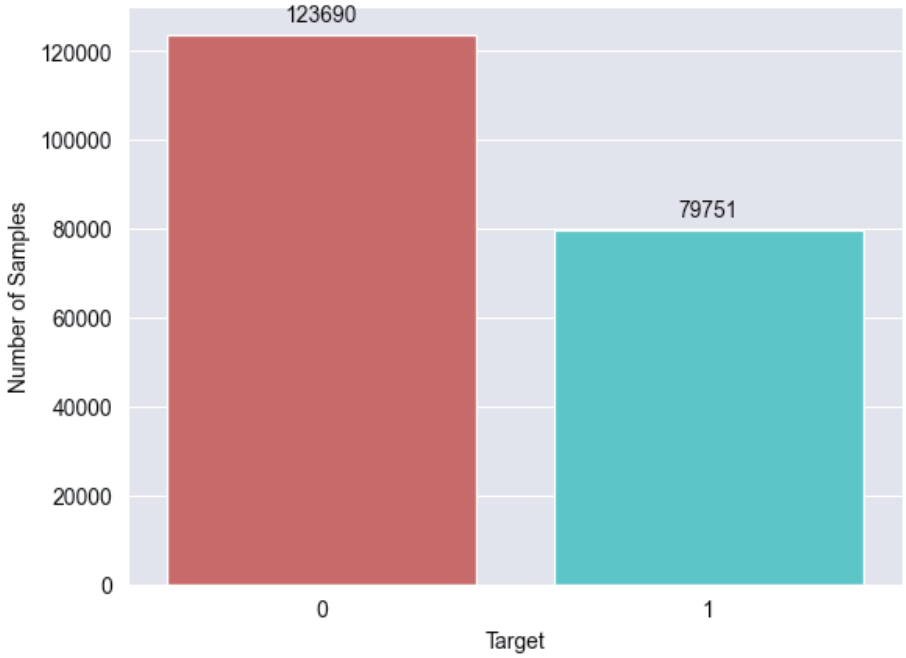

Figure 3.2: Target Distribution of the Merged Dataset

The distribution of labels in this dataset is presented in Figure 3.2, showing that 60.80% of the data, or 123,690 rows, are normal data. The remaining 39.20%, or 79,751 rows, represent attack data. To accurately mirror the data distribution in real-world scenarios and avoid bias caused by data balancing, this dataset is kept unbalanced.

3.1.2 Dataset Splitting

In this thesis, the proposed framework contains a server and multiple clients. A dataset split that reflects this structure is required, beyond merely dividing into training and testing sets. To demonstrate and test the proposed framework, we use three clients as an example. The process of dataset splitting is illustrated in Figure 3.3. Initially, the merged dataset is divided into a training set and a testing set, with a ratio of 80% and 20%, respectively.

Considering that the merged dataset includes both sensing data and network traffic data, it is essential to split the dataset in alignment with the distinct functions

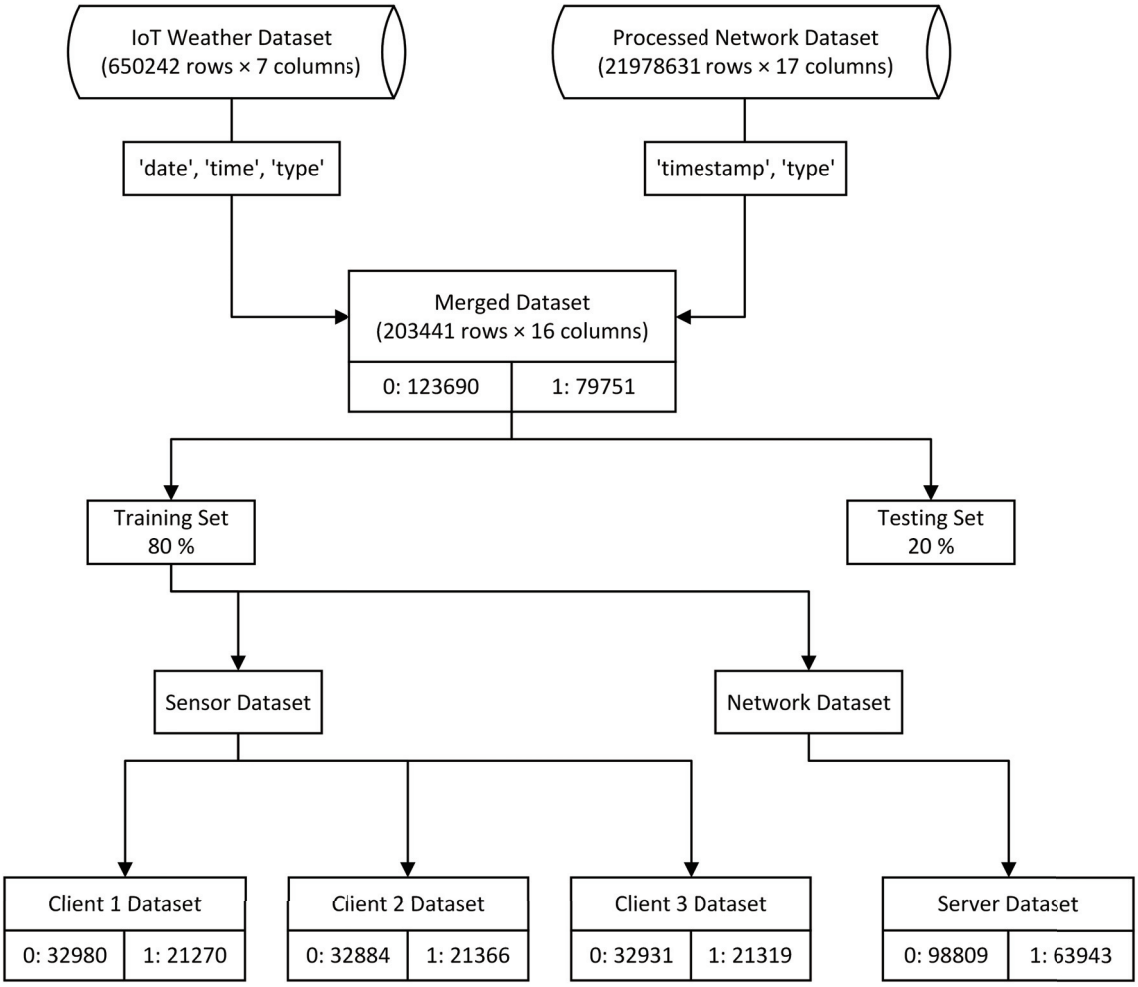

Figure 3.3: Dataset Processing and Splitting

of the server and the clients. Specifically, the server requires only network traffic data, while the clients require only sensing data. Therefore, the training set is further divided based on data type, ensuring the label column is retained in both subsets. The obtained datasets are classified as the sensor dataset and the network dataset.

For this experiment, the sensor dataset is evenly split among the three clients, with each receiving over fifty thousand data entries. The server uses the entire network dataset, amounting to 162,752 entries, which is equivalent to the combined data quantity allocated to the three clients. Through these steps, four distinct datasets are generated: the network dataset for the server, and three sensor datasets for the clients.

3.2 Proposed Framework

In large WSNs, multiple sink nodes are tasked with data collection from assigned wireless sensor nodes. These networks use a gateway that serves as an intermediary between the sink nodes and the Internet. This gateway's primary functions include communication with each sink node and the aggregation and processing of data. This described architecture shares similarities with the global-local model structure in Federated Learning (FL), as applied within the Internet of Things (IoT) context. Therefore, this thesis proposed a framework for WSNs, a hybrid ML IDS with integrated server and client models, named Server-Client Machine Learning Intrusion Detection System (SC-MLIDS).

3.2.1 Architecture

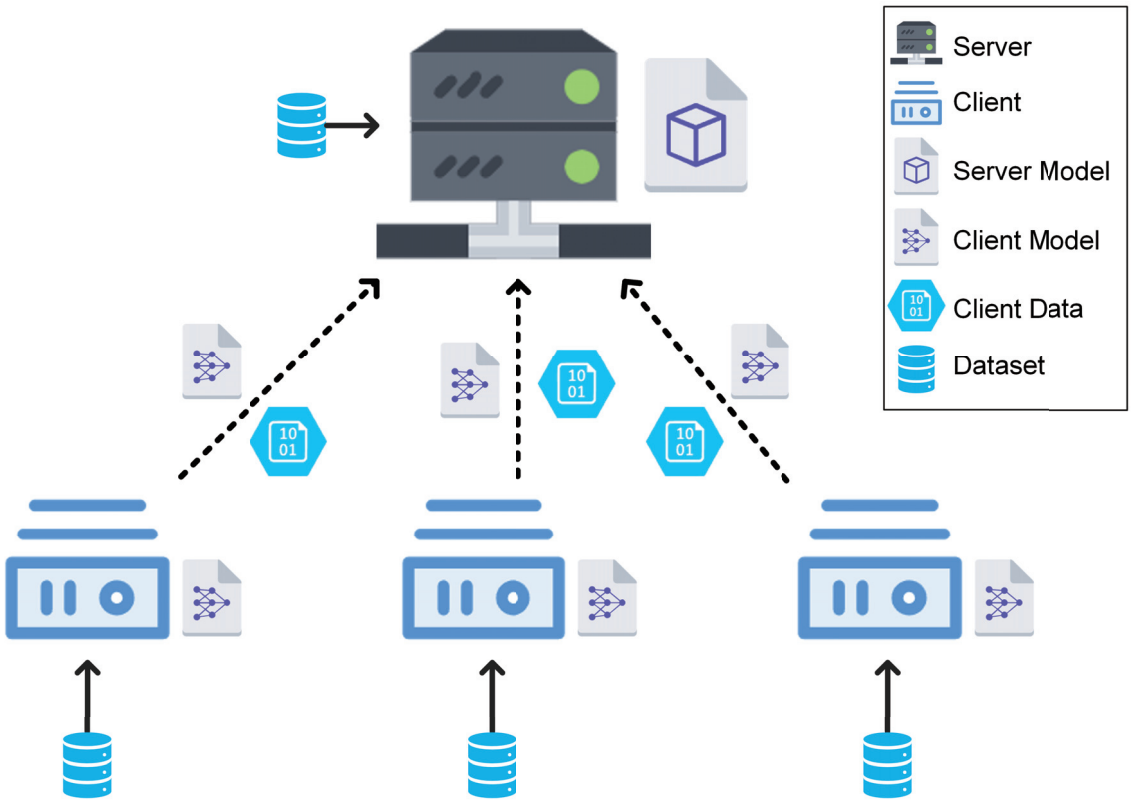

Figure 3.4: Proposed Framework

The SC-MLIDS framework, as shown in Figure 3.4, contains two primary components: a server and multiple clients.

- **Server:** Represents the gateway in the WSN.
- **Client:** Represents the sink nodes in the WSN.
- **Client Model:** This component is responsible for validating sensing data. Such data are generated and transmitted by wireless sensor nodes that are managed by the sink node.
- **Server Model:** This component is responsible for validating network traffic data. This network traffic is associated with the sensing data that the gateway receives from the sink node.
- **Client Data:** The sensing data that collected by the sink node from its administered wireless sensor nodes and filtered by the client model.
- **Dataset:** A collection of sensing data matched with corresponding network traffic data. This dataset is used to train both the client and server models.

During the WSN's initial deployment phase, it is crucial to prepare a dataset containing both sensing data and the corresponding network traffic data. This dataset is used for training the ML models on both the clients and the server. The dataset should be partitioned based on the number of sink nodes, according to the method delineated in Figure 3.3.

The initial deployment phase of SC-MLIDS includes seven steps:

1. **Distribute the Split Sensor Dataset:** Each client receives a portion of the split sensor dataset.
2. **Train Client Models:** In each client, a client model is trained using the assigned sensor dataset.
3. **Compress and Encrypt Client Model Files:** After training, the client model files are compressed and encrypted for secure transmission.

4. **Transmit Client Model Files to the Server:** These compressed and encrypted client model files are then sent to the server.

5. **Server Reception and Processing:** Upon receipt, the server decompresses and decrypts all the client model files.

6. **Server Model Training:** The server trains a server model using the network traffic dataset.

7. **Model Aggregation:** The server aggregates all the client models and the server model using the model aggregation algorithms.

During the formal operational phase, the SC-MLIDS framework follows a six-step operational process:

1. **Data Collection by Sensor Nodes:** These nodes actively sense environmental conditions, generating relevant sensing data.

2. **Data Transmission to Sink Nodes:** The sensor nodes transmit the sensing data to their affiliated sink nodes.

3. **Data Validation at Sink Nodes:** Upon receipt, the sink nodes employ the integrated client model to validate the received sensing data.

4. **Sending Data to the Gateway:** The sink nodes forward the validated sensing data, along with the corresponding network traffic data, to the gateway.

5. **Gateway-Level Validation:** The gateway applies aggregation prediction algorithms, in conjunction with the client and server models, to conduct a final validation of both the sensing data and the corresponding network traffic data.

6. **Uploading Server-Validated Data:** Following validation, the server-validated data is uploaded to the Internet for additional analysis and processing by end-users.

The proposed SC-MLIDS framework is designed to implement two-layer intrusion detection, targeting both the sink node and gateway levels. This approach is

particularly applicable considering the inherent vulnerabilities of WSN nodes to network attacks, largely due to their limited computational capacity for deploying robust intrusion-resistant mechanisms.

Common attacks on WSNs primarily involve damage to nodes and network functions or data contamination.

At the sensor node level, a compromised node poses a significant risk. The sensing data generated by such a node can be subject to tampering, leading to erroneous information being introduced into the network. This compromised data, once integrated into the network, can severely undermine the reliability of the entire database, thereby devaluing its overall integrity. Consequently, the SC-MLIDS framework prioritizes the verification of data authenticity during the data collection phase, yielding three key advantages:

1. **Authenticity Verification:** This process ensures a preliminary assessment of the security status of the sensor nodes, as the data they generate is authenticated at the outset.

2. **Reliability Assurance:** By verifying the sensing data, the possibility of the database being contaminated with malicious data is significantly reduced, thereby ensuring its reliability.

3. **Efficient Data Management:** The framework facilitates the filtering of sensing data, and suspicious data are discarded. This not only saves network and computational resources by reducing the volume of data needing transmission but also alleviates some of the verification workload on the server side.

Overall, the SC-MLIDS framework is strategically positioned to enhance security and efficiency in WSN operations, addressing critical vulnerabilities and optimizing resource utilization.

At the gateway level, the SC-MLIDS framework uses server models that have been trained on network traffic data. This training enables the identification of common network attacks through traffic analysis, thereby facilitating effective intrusion detection within WSNs. The integration of various client models at the gateway further enhances security.

The use of aggregation prediction algorithms allows for comprehensive validation of data from sensor nodes within the administration of a specific sink node. Since the sensing data arriving at the gateway has already experienced initial validation by the client model at the respective sink node, the subsequent extensive validation by all models at the gateway level significantly enhances both data reliability and network security.

Regarding the client models within the SC-MLIDS framework, they must be pre-trained. During this phase, data encryption and compression algorithms should be configured to meet the specific requirements of the WSN application. The pre-configuration of encryption keys or seeds, along with salts, is crucial for reducing the risks of data leakage during network transmission.

Furthermore, the SC-MLIDS framework demonstrates flexibility in its applicability to small or simple WSNs. In such configurations, the structure is simplified to include just a server and a client. Here, the sensor node gathers sensing data and forwards it to the sink node for initial validation. The sink node then transmits this validated data to the gateway, where comprehensive validation of both the sensing data and network traffic data is conducted. The essential models for the aggregation prediction algorithms in this scenario include a client model and a server model. This simplified structure effectively maintains the two-layer intrusion detection characteristic of the SC-MLIDS framework, thereby ensuring robust security even in less complex WSN configurations.

3.2.2 Aggregation Prediction Algorithms

The SC-MLIDS framework employs model aggregation methodologies at the gateway level, similar to the bagging approach in Ensemble Learning (EL). This process includes two aggregation prediction algorithms:

1. **Weighted Score Algorithm:** This algorithm combines the prediction results of each model based on their performance metrics and assigned weights. The resulting scores are then converted into binary classification results, ensuring holistic and comprehensive prediction results.

2. **Majority Voting Algorithm:** This algorithm operates on a majority voting

principle, where the predictions made by each model are collectively analyzed. The final prediction is then determined based on the results that receive the majority votes among these models.

Due to the flexibility of the proposed two aggregation prediction algorithms, models trained using a variety of classifiers can also be aggregated effectively. This holds regardless of the type and number of classifiers employed. Additionally, the strategy of employing a combination of diverse classifiers, based on specific WSN application scenarios and data characteristics, is also feasible and supported.

These algorithms enhance the robustness and accuracy of the SC-MLIDS framework by using the diverse strengths of the individual models involved. The use of both weighted score and majority voting provides a comprehensive approach to model prediction, ensuring more reliable and effective intrusion detection in WSNs.

Considering that both algorithms perform the same task, it is crucial to choose a suitable algorithm. Therefore, it is necessary to examine their performance based on WSN application scenarios and special requirements and select the appropriate algorithm that best suits the current needs. This will ensure that the SC-MLIDS framework achieves optimal intrusion detection in different WSN environments.

Weighted Score Algorithm

Given the unique characteristics of WSNs, Precision emerges as a particularly crucial metric in assessing the performance of ML models used for such networks. In WSNs, the authenticity and accuracy of data are crucial for subsequent data analysis processes. Consequently, ML models for WSN applications should prioritize minimizing false positives. This focus on Precision reflects the lower relative significance of false negatives in this context.

Precision is defined as the proportion of all samples predicted to be positive that are actually positive. A higher Precision indicates that the model performs greater caution in its predictions, effectively reducing the occurrence of false positives. Additionally, Recall or True Positive Rate (TPR) denotes the proportion of actual positive samples correctly identified as positive by the model. The F1 Score addresses potential imbalances within the dataset by harmonizing Precision and Recall. This score is particularly relevant when dealing with unbalanced datasets.

Therefore, while Precision remains a critical metric for the performance evaluation of ML models in WSNs, the F1 Score also deserves consideration, especially in scenarios involving dataset imbalance.

The weighted score algorithm proposed in this thesis innovatively integrates model performance metrics into the bagging methodology of EL, designed specifically for the characteristics of WSNs. As presented in Algorithm 1, this algorithm synthesizes the final prediction results by considering a range of inputs. These inputs include the individual prediction results of each model, the Precision and F1 Score of these models, their respective weight shares, and the overall weight assigned to each model.

This approach allows for a precise final prediction, which is informed not only by the predictive results of the individual models but also by their respective performance metrics and assigned weights. By incorporating both Precision and F1 Score into the weighting mechanism, the algorithm effectively acknowledges the importance of precision and balanced performance in the WSNs context. Therefore, this method offers a more comprehensive and contextually appropriate approach to ML models aggregation, enhancing the reliability and applicability of the predictions in WSN environments.

In the proposed weighted score algorithm, the initial step involves running each client model and the server model to generate their respective independent predictions. These predictions are stored in a two-dimensional array format. For instance, the array might look like [[0, 0, 0, ..., 1], [1, 0, 0, ..., 1], [1, 0, 0, ..., 1], ..., [0, 1, 1, ..., 1]], where the last array corresponds to the prediction result of the server model.

Following this, the Precision and F1 Score obtained during the testing phase of each model are provided and also stored as two-dimensional arrays. An example of such an array could be [[0.9746, 0.9927], [0.9615, 0.7805], [0.8183, 0.6399], ..., [0.9434, 0.9337]].

Subsequently, the weights are assigned to each model. These weights are represented as an array, such as [0.1, 0.1, 0.1, ..., 0.5], with the condition that their cumulative sum equals 1. This array configuration is crucial as it reflects the relative importance and performance of each model within the aggregation prediction process. For instance, in the SC-MLIDS framework, the server model is typically assigned a greater weight compared to client models. This is due to the server model being

Algorithm 1: Aggregation Prediction Algorithm by Weighted Score

Input : *models_predictions*: 2D array, predictions of all models.
models_metrics: 2D array, Precision and F1 of all models.
models_weights: Array, the weights of each model.
precision_weight: Floating point, the precision weight (default 0.6).
f1_weight: Floating point, the f1 weight (default 0.4).

Output: *final_predictions*: Array of final predictions.

```
weighted_scores = [ ]
for m in models_metrics do
    score ← m[0] × precision_weight + m[1] × f1_weight
    weighted_scores.append(score)
end for
weighted_predictions = [ ]
for i ← 0 to length of models_predictions - 1 do
    weighted_pred ←
     models_predictions[i] × models_weights[i] × weighted_scores[i]
    weighted_predictions.append(weighted_pred)
end for
results ← sum of weighted_predictions / sum of
  (models_weights × weighted_scores)
threshold ← 0.5
if results ≥ threshold then
    final_prediction ← 1
else
    final_prediction ← 0
end if
return final_predictions
```

trained using network traffic data, which often makes it more critical in the overall predictive accuracy.

Finally, the algorithm requires the assignment of weights to the Precision and F1 Score. The weight is determined based on the characteristics of the dataset used to train the models. For example, the weights might be assigned as 0.6 and 0.4 for Precision and F1 Score, respectively. This weighting approach allows for a balance between Precision and F1 Score, ensuring that the aggregation prediction is not only accurate but also sensitive to the dataset's unique properties.

After completing the preparation of all required inputs, the weighted score for each model is calculated as a linear combination of its Precision and F1 Score.

Let P_i and F_i represent the Precision and F1 Score for the i^{th} model, respectively. The weighted score S_i is then given by:

$$S_i = \alpha \cdot P_i + (1 - \alpha) \cdot F_i \tag{3.1}$$

where α is a user-defined parameter determining the relative importance of Precision versus F1 Score.

Let $pred_i$ represent the array of binary predictions from the i^{th} model, and w_i be its assigned weight. The adjusted predictions adj_pred_i for each model are calculated as:

$$adj\_pred_i = pred_i \cdot w_i \cdot S_i \tag{3.2}$$

This adjustment scales the predictions of each model according to both its performance metrics and its assigned weight.

The final step involves aggregating these adjusted predictions across all models and normalizing the result. Let N be the number of models. The aggregated prediction A is given by:

$$A = \frac{\sum_{i=1}^{N} adj\_pred_i}{\sum_{i=1}^{N} w_i \cdot S_i} \tag{3.3}$$

This equation computes a weighted average of the adjusted predictions, normalized by the total weight-adjusted scores of all models.

Finally, a threshold θ (commonly 0.5 for binary classification) is applied to A to obtain the final binary classification P:

$$P_j = \begin{cases} 1 & \text{if } A_j \geq \theta \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

for each element j in A.

After the above operational steps, the result given by the weighted score algorithm will be an array storing the final predictions. For instance, the array might look like [0, 1, 1, ..., 0].

In summary, the proposed weighted score algorithm offers a comprehensive approach for scoring model performance and weights in WSNs. This is achieved by considering various factors: the individual prediction results of each model, the performance metrics of the models, the weights assigned to each model, and the assignment of weights among these metrics. A notable aspect of this algorithm is that the final prediction tends to be more closely aligned with models assigned higher weights. Furthermore, if a model with a high weight also gives high-performance metrics, the algorithm's predictions are possible to closely mirror the metrics of that model. However, it is important to acknowledge that due to the aggregate nature of the predictions, there may be cases where the algorithm's performance metrics are slightly lower than those of the best-performing individual model.

The proposed weighted score algorithm is specifically designed for WSNs. By incorporating both Precision and F1 Score, along with their respective weights, the algorithm is highly applicable to WSN environments where minimizing false positives is a critical concern. Despite the applicability of this algorithm is not limited to WSNs. With appropriate modifications, such as the integration of additional model metrics or alterations of Precision and F1 Score, the algorithm can be adapted to other application scenarios requiring model aggregation prediction.

Therefore, in this thesis, the algorithm has been optimized for WSNs to accommodate their specific characteristics and needs. As an indispensable component of the SC-MLIDS framework, it presents a robust and effective method for model aggregation prediction, enhancing the performance of two-layer IDSs in WSNs. This adaptability and precision emphasize the potential of the algorithm as an option in various model aggregation scenarios.

Majority Voting Algorithm

Majority voting, a frequently employed method in EL, operates on the principle of using predictions from multiple models, with the class receiving the majority of votes

being selected as the final predictions. However, considering the specificities of the SC-MLIDS framework, especially within the context of WSNs, a direct application of majority voting is not feasible and requires modification to align with the server-client architecture.

In WSNs applying the SC-MLIDS framework, the diversity of training sets and the limited data collection scope of individual client models at each sink node may not fully cover the range of potential scenarios in data collection. Consequently, an aggregated prediction using the client models from each sink node, combined with majority voting, yields more accurate predictions. In addition, the server model, trained on a comprehensive network traffic dataset, tends to have superior performance in detecting various network traffic.

To accommodate these considerations, this thesis proposed a modified approach of partial majority voting, as presented in Algorithm 2. In this algorithm, majority voting is primarily based on the prediction results of the client models, but the server model's predictions carry a decisive weight and influence due to its comprehensive training and superior performance.

In the proposed majority voting algorithm, each client and server model independently generates prediction results. These are stored as a two-dimensional array, for example, [[0, 0, 0, ..., 1], [1, 0, 0, ..., 1], [1, 0, 0, ..., 1], ..., [0, 1, 1, ..., 1]], where the final array represents the server model's predictions.

Upon inputting the prediction results from each model into the majority voting algorithm, the algorithm proceeds to compute the predictions made by each model. Since this algorithm primarily addresses binary classification models, the prediction results can be straightforwardly computed. For instance, in the case of three client models, the summation of predictions for a specific sample could yield counts such as 0, 1, 2, or 3. These numbers represent the frequency with which the sample is predicted as the negative class (denoted as 1) across the three models.

To account for potential errors and ensure robustness, the majority voting algorithm adopts a threshold for majority determination. Particularly, a prediction result is considered a majority if it is indicated as a negative class in more than two-thirds of the client models. The majority voting results are then matched with the predictions made by the server model. If there is a consensus between the majority voting

Algorithm 2: Aggregation Prediction Algorithm by Majority Voting

Input : *models_predictions*: 2D array, predictions of all models.

Output: *final_predictions*: Array of final predictions.

```
majority_votes = [ ]
for i ← 0 to length of models_predictions[0] do
    result_sum ← 0
    for j ← 0 to length of models_predictions - 1 do
        result_sum ← result_sum + models_predictions[j][i]
    end for
    threshold ← 2/3 (length of models_predictions - 1)
    if result_sum ≥ threshold then
        vote ← 1
    else
        vote ← 0
    end if
    majority_votes.append(vote)
end for
final_predictions = [ ]
for k ← 0 to length of majority_votes do
    if majority_votes[k] = models_predictions[−1][k] then
        final_predictions.append(majority_votes[k])
    else
        final_predictions.append(1)
    end if
end for
return final_predictions
```

result and the server model's prediction, this result is denoted as the final prediction. Otherwise, the algorithm defaults to recording the prediction as a negative class.

Specifically, for each prediction sample i, the algorithm sums up the predictions from all models except the last one (server model). Let A_{ji} denote the prediction of the j^{th} model for the i^{th} sample. The sum S_i for each sample is given by:

$$S_i = \sum_{j=1}^{N-1} A_{ji} \tag{3.5}$$

where N is the total number of models, and $N-1$ represents all models except the last one.

The algorithm then applies a threshold θ (generally two-thirds of the number of client models) to this sum to determine the majority vote V_i for each sample:

$$V_i = \begin{cases} 1 & \text{if } S_i \geq \theta \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

This reflects the rule that if more than two-thirds of models predict as 1, the majority vote is 1; otherwise, it's 0.

The final prediction P_i for each sample is determined by comparing the majority vote with the prediction of the last model:

$$P_i = \begin{cases} V_i & \text{if } V_i = A_{Ni} \\ 1 & \text{otherwise} \end{cases} \tag{3.7}$$

Here, A_{Ni} is the prediction of the last model for the i^{th} sample. If the majority vote matches the last model's prediction, it is taken as the final prediction; otherwise, the final prediction defaults to 1.

The proposed majority voting algorithm eventually produces an array that represents the final prediction, such as [0, 1, 1, ..., 0]. This array is the result of the majority voting algorithm produced, which synthesizes the predictions from each model.

Distinct from general majority voting and weighted majority voting approaches, the proposed majority voting algorithm uniquely assigns a decisive voting weight to the model showing superior performance. This customized approach aligns with the server-client structure inherent in the SC-MLIDS framework. It ensures that the

predictions made by the client model are co-verified by the server model. The jointly verified results from these two model types are then adopted as the final prediction results. This approach is particularly effective in WSNs, where it is crucial to minimize false positives in data.

The proposed majority voting algorithm, as configured in this thesis, is predisposed to yield results leaning toward the negative class. This bias is intentional, deriving from the algorithm's design where discrepancies between the two types of model predictions default to the negative class, thereby reducing the occurrence of false positives. However, this configuration is not rigid; the algorithm's matching conditions and majority voting thresholds can be customized to suit various application scenarios. For instance, adjusting the decisive role's voting weight or altering the majority voting threshold can modify the prediction result's tolerance.

In conclusion, this thesis has optimized the majority voting algorithm to address the specific needs of WSNs, particularly their emphasis on minimizing false positives. As an integral part of the SC-MLIDS framework, this algorithm presents an additional secure and effective method for model aggregation prediction, demonstrating its adaptability and utility in diverse scenarios beyond WSNs.

3.3 Implementation Details

In this thesis, we have proposed the SC-MLIDS, a novel hybrid ML IDS framework designed for WSNs. The SC-MLIDS framework uses a server-client structure in which data from sensor nodes within the WSN is initially validated at the sink node using the client model. Subsequently, sensing data that successfully pass this initial validation is transmitted to the gateway. Concurrently, network traffic data generated during this transmission is also collected. At the gateway, both the sensing data and the corresponding network traffic data experience further validation through aggregation prediction algorithms that integrate the client and server models.

To demonstrate the practical viability of the proposed SC-MLIDS framework, we have developed a simulation program, which has been implemented using Python. This program, along with the datasets, demonstrations, detailed descriptions, and other relevant materials, has been made publicly available. These resources can be

accessed via the GitHub repository named *SC-MLIDS*[1]. This repository has been established to facilitate easy access, replication, and further exploration by researchers in the field.

In this thesis, we have developed a Python-based simulation program to simulate the communication between sink nodes and the gateway in WSNs. This program is partitioned into four modules: Utilities, Server, Client, and Aggregation Prediction. Each module's code and functionality are elaborated as follows:

- **Utility Module:** This module serves as an auxiliary tool within the simulation program. It contains a variety of functions, including importing and splitting datasets, outputting distributions of dataset labels, generating model performance metrics, and creating encryption keys.

- **Server Module:** Designed to simulate the gateway's role in a WSN, this module handles communication with clients. It manages the reception of client messages, including metadata, ML model files, and subsequent sensing and network traffic data. Additional functionalities include the decryption and decompression of model files and the training of the server model.

- **Client Module:** Simulating the sink node's function in a WSN, this module facilitates communication with the server. The client's primary tasks include training the client model, compressing and encrypting the model, and then transmitting it to the server. The module also handles the transmission of sensing data to the server. In scenarios involving multiple clients, the client module can be duplicated and modified with distinct client IDs to simulate various client entities.

- **Aggregation Prediction Module:** This module implements the two aggregation prediction algorithms specifically designed for the SC-MLIDS framework. Additionally, it includes a helper method for integrating and generating the necessary inputs for these algorithms and for displaying the performance metrics of the models.

[1]https://github.com/Hongwei-Z/SC-MLIDS

In our simulation, the SC-MLIDS framework's functionality is showcased during both the initial deployment phase and the formal operational phase of WSNs. During the initial deployment, the program executes model training, testing, and transmission. In the operational phase, it manages sensing data collection, validation, transmission, file processing, and aggregated validation.

To simulate the SC-MLIDS framework in the WSN's initial deployment phase, the program operates as follows:

1. **Data Preparation:** The processed TON_IoT dataset is divided into a training set and a testing set. The training set is further split into subsets for sensing data and network traffic data, with the sensing data subset being divided again based on the number of clients.

2. **Client Model Training:** Each client model is trained using its respective subset of sensing data, using the Random Forest (RF) classifier from the 'scikit-learn' library.

3. **File Saving:** The trained client model is then saved as a file using a method provided by the 'joblib' library.

4. **File Encryption:** A seed and salt are generated to create a key, which is used to encrypt the model file using the Fernet algorithm from the 'cryptography' library, based on Advanced Encryption Standard (AES) symmetric encryption.

5. **File Compression:** The 'zlib' library's file compression method is applied to the encrypted client model file.

6. **Client-Server Connection Establishment:** The 'socket' library is used to achieve client-server connection by defining the host and port.

7. **File Transmission:** The clients send metadata (including seed, salt, and file size) to the server, followed by the encrypted and compressed model file.

8. **Server-Client Connection:** The server connects with each client.

9. **File Reception and Verification:** The server receives metadata and model files from each client, verifying the file sizes to ensure completeness.

10. **File Decompression and Decryption:** Each received model file is decompressed and decrypted by the server and then serialized into a file.

11. **Reception and Processing Repetition:** Steps 9 and 10 are repeated until all client model files are received and processed.

12. **Server Model Training:** The server trains its model using the network traffic data subset and serializes this server model into a file.

In the operational phase simulation of the SC-MLIDS framework within a WSN, the program iteratively executes the following steps:

1. **Server-Client Communication:** The server establishes and maintains communication with each client.

2. **Client-Level Data Validation:** Sensing data is input into the client and experiences validation via the client model.

3. **Data Transmission to Server:** Sensing data that successfully passes the validation at the client level is then transmitted to the server. Conversely, data failing this validation is discarded.

4. **Server-Side Data Integration:** Upon receipt, the server integrates the sensing data with the corresponding network traffic data generated during communication.

5. **Server-Level Data Validation:** The server then employs aggregation prediction algorithms for a comprehensive evaluation of the merged data.

6. **Data Preservation and Discard:** Data that successfully pass validation on the server side is preserved, while data failing to pass this validation is discarded.

The detailed step-by-step description of the simulation for both the initial deployment and operational phases elaborates on the functioning of our developed simulation program. This substantiates the practical viability of the proposed SC-MLIDS framework. Notably, in our simulation program, RF classifiers are uniformly used at both the client and server levels. The choice of RF is driven by their typically high

accuracy, robust performance, and efficiency, particularly with unbalanced datasets and a multitude of predictive variables [76, 77, 78]. Furthermore, their widespread support across numerous open-source libraries facilitates ease of use and avoids the need for extensive parameter tuning.

Given these advantages, the RF classifier is consistently used throughout our simulation. However, it is critical to note that the SC-MLIDS framework and its integrated model aggregation prediction algorithms are not restricted to this single classifier type. The framework is flexible, supporting the integration of various other classifiers depending on specific application scenarios and data characteristics. Moreover, due to the adaptability of the proposed aggregation prediction algorithms, models developed using diverse classifiers can be effectively aggregated, independent of their type and quantity.

Chapter 4

Experimental Setup and Results

4.1 Experimental Setup

To conduct a comprehensive evaluation of our proposed Server-Client Machine Learning Intrusion Detection System (SC-MLIDS) framework, we developed a customized simulation program. This program contains key functions such as facilitating communication between the server and client, training the models, and executing the aggregation prediction algorithms. Through the deployment of this simulation program, a series of experiments were implemented.

The experimental phase was executed on a high-performance laptop. This laptop is equipped with an Intel Core i9-13900H CPU and an NVIDIA GeForce RTX 4060 Laptop GPU, offering advanced computing capabilities. The detailed specifications of this laptop are provided in Table 4.1.

Table 4.1: Testing Platform

Laptop	Lenovo ThinkBook 16p Gen 4
CPU	Intel Core i9-13900H 2.60 GHz
Graphics Card	NVIDIA GeForce RTX 4060 Laptop GPU 8 GB
Memory	Samsung 32 GB (16 GB × 2) DDR5 5200 MHz
Storage	Samsung PM9A1 1 TB PCIe 4.0 × 4 M.2 SSD
Operating System	Microsoft Windows 11 Home 64-bit

To guarantee both the flexibility and reproducibility of our experiments, Python was chosen as the primary programming language. The software environment for these experiments is Python 3.10, with critical libraries including 'scikit-learn', 'socket', 'joblib', 'matplotlib', 'pandas', and 'numpy'.

The core aim of our simulation program is to demonstrate client-server interaction when SC-MLIDS framework is applied to a Wireless Sensor Network (WSN) environment. We use the collected sensing data to train the client model at the client. At

the server, we use network traffic data for training the server model and implementing the model aggregation. To maintain data integrity and security throughout this process, the client models are compressed and encrypted before their transmission to the server.

To simulate local communication on the laptop, the host and port numbers for both the client and server were uniformly set to '127.0.0.1' and '8080', respectively. This experimental setup included one server and three clients. The processed dataset was divided such that 80% was allocated as the training set and the remaining 20% as the testing set. Each client was assigned one-third of the training set, while the server used the entirety of this set.

In terms of evaluation, each model's performance was assessed using metrics such as Accuracy, Precision, Recall, F1 Score, and the Receiver Operating Characteristic (ROC) curves. Additionally, we implemented the two proposed model aggregation prediction algorithms to merge the prediction results of these models. This approach aimed to enhance the overall detection efficiency and accuracy, yielding comprehensive prediction results that are not solely dependent on a single model's performance.

The proposed weighted score algorithm is based on model performance and weights, the weights assigned to the three client models were set uniformly at 0.2 each, while the server model was assigned a higher weight of 0.4, reflecting its greater importance and performance. In terms of performance metrics, particular attention was given to the Precision and F1 Score. The weight assigned to Precision was set at 0.6, while the F1 Score was weighted at 0.4. This specific weighting strategy was adopted because we emphasized minimizing false positives in the model predictions. Additionally, it reflects a balanced trade-off in the model's predictive accuracy, considering both the precision and the holistic performance as represented by the F1 Score.

4.2 Performance Metrics

In this thesis, the performance of the Machine Learning (ML) models and the aggregation prediction algorithms are evaluated using four key metrics: Accuracy, Precision, Recall, and F1 Score. The formulas for these metrics are as follows:

$$TP = \text{True Positives}$$
$$FP = \text{False Positives}$$
$$TN = \text{True Negatives}$$
$$FN = \text{False Negatives}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$
$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$
$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$
$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4.4}$$

- **Accuracy:** This metric quantifies the proportion of correctly predicted classes by the model. It is calculated by dividing the number of correctly predicted samples by the total number of samples. Accuracy serves as a fundamental metric for assessing the performance of ML models.

- **Precision:** Precision measures the proportion of actual positive classes among those predicted as positive by the model. It is calculated by dividing the number of samples correctly predicted as positive by the total number of samples predicted as positive. Precision is particularly critical in scenarios involving unbalanced datasets.

- **Recall:** Recall calculates the ratio of samples correctly identified as positive by the model to the total number of actual positive samples. This metric reflects the model's capability to correctly identify positive classes. Recall is also known as True Positive Rate (TPR).

- **F1 Score:** The F1 Score provides a harmonized metric that balances Precision and Recall. It is especially useful for obtaining a comprehensive view of the model's performance, particularly when dealing with unbalanced datasets.

These metrics were selected not only to fulfill the basic requirements for evaluating ML models but also to cater specifically to the needs of WSNs, ensuring a thorough and relevant assessment of model performance.

4.3 Results

The proposed SC-MLIDS framework is characterized by two key software components: server-client communication and model aggregation prediction algorithms. To facilitate a comprehensive evaluation of the SC-MLIDS framework, our experimental design was separated into two distinct parts, corresponding to the two software components.

4.3.1 Server-Client Communication

In our experimental evaluation of the SC-MLIDS framework, we simulated and analyzed the communication process between the server and three clients using a developed simulation program.

Table 4.2: Model Training and Transmission Results

Participants	**Server**	**Client 1**	**Client 2**	**Client 3**
Address	127.0.0.1 8080	127.0.0.1 54742	127.0.0.1 54814	127.0.0.1 54830
Label Distribution	0: 98809 1: 63943	0: 32980 1: 21270	0: 32884 1: 21366	0: 32931 1: 21319
Model Training Time (seconds)	6.8305	8.9069	8.3103	8.1050
Model File Size (MB)	18.7705	106.4463	106.6699	106.2451
Encrypted Size (MB)	N/A	141.9277	142.2266	141.6600
Encryption Time (seconds)	N/A	0.7266	0.5947	0.5654
Compressed Size (MB)	N/A	107.5145	107.7409	107.3119
Compression Time (seconds)	N/A	5.2522	3.8984	3.8393
Sending Time (seconds)	N/A	0.0100	0.0080	0.0080
Received File Size (MB)	N/A	107.5145	107.7409	107.3119
Receiving Time (seconds)	N/A	80.2155	64.8709	76.1133
Average Rate (MB/s)	N/A	1.3403	1.6609	1.4099
Decompression & Decryption Time (seconds)	N/A	1.1003	0.8955	1.1386

Table 4.2 systematically presents and summarizes this process. Each client model

is trained using sensing data, then encrypted and compressed before being sent to the server. The server, upon receiving these model files, proceeds to decompress and decrypt them. Subsequently, the server model is trained using network traffic data. To ensure uniformity across the experiment, all models uniformly employed the Random Forest (RF) classifier.

For this local transmission simulation, clients were configured with loopback addresses but with distinct ports: Client 1 (127.0.0.1, 54742), Client 2 (127.0.0.1, 54814), and Client 3 (127.0.0.1, 54830). Each client was assigned a training set of roughly equivalent size, though variations in label distribution were observed. Specifically, the label distribution for Client 1 was 0: 32980, 1: 21270; for Client 2 was 0: 32884, 1: 21366; and for Client 3, it was 0: 32931, 1: 21319.

The model training time for all three clients averaged just over 8 seconds. In contrast, the server, processing three times the amount of data, completed its model training in merely 6.83 seconds. The server model file, not being transmitted, was exempt from encryption and compression procedures. The generated model files across the clients were approximately 106 MB in size. Encryption of these files was fast, ranging from 0.57 to 0.73 seconds, but resulting in a file size increase to about 142 MB. After compression, the model files were approximately 107 MB, slightly larger than the original size, with compression times varying from 3.84 to 5.25 seconds for the clients.

The transfer of the encrypted and compressed model files to the server was executed with remarkable speed, showcasing the efficiency of the communication process. The consistency in the size of the files received by the server compared to those sent by the clients confirmed the integrity of the data during transmission. However, notable variations were observed in the reception times. Specifically, Client 1 experienced a longer reception time of approximately 80 seconds, while Client 2 completed the transfer the quickest, in about 65 seconds, and Client 3 fell in between with a duration of 76 seconds.

The average transmission rate for all clients exceeded 1 MB/s, with Client 2 achieving the highest rate at 1.66 MB/s. During the final phases of decompression and decryption, each client completed the task in approximately 1 second, further demonstrating the framework's effectiveness in secure model transmission.

In a subsequent round of experiments, we employed different classifiers for training the server and client models. Partial results are shown in Table 4.3. This included a selection of popular classifiers such as RF, Logistic Regression (LR), Gradient Boosting (GB), and Support Vector Machine (SVM).

Table 4.3: Model Training Results Using Different Classifiers

Participants	**Server**	**Client 1**	**Client 2**	**Client 3**
Classifier	Support Vector Machine	Random Forest	Logistic Regression	Gradient Boosting
Label Distribution	0: 98809 1: 63943	0: 32980 1: 21270	0: 32884 1: 21366	0: 32931 1: 21319
Training Time (seconds)	90.806	8.622	0.206	4.672
File Size (MB)	1.3936	108.0674	0.0020	0.1836

Despite using the same training sets, there was a marked difference in the file sizes of the models generated by the different classifiers. Client 1, using the RF classifier, produced a model file size similar to the previous round, approximately 108 MB. Client 2, using LR, generated a significantly smaller model file of only 2 KB. Client 3, using GB, produced a model file of 188 KB. The server model trained using SVM, resulted in a file size of only 1.39 MB. These results highlight that the choice of classifier not only impacts the size of the resulting model file but also influences the efficiency of transmission, thereby affecting the overall performance.

This experiment underscores the SC-MLIDS framework's capacity to facilitate secure and efficient model training and transfer between the server and clients. Such hybrid ML showcases the potential of the SC-MLIDS framework in intrusion detection.

4.3.2 Aggregation Prediction Algorithms

To thoroughly evaluate the model aggregation prediction algorithms within the proposed SC-MLIDS framework, we initially assessed the independent performance of each model using the testing set. This assessment involved deriving metrics such as Accuracy, Precision, Recall, F1 Score, and the ROC Curve.

As shown in Table 4.4, the performance metrics of the RF-based models reveal

Table 4.4: Simulation Results without Applying SC-MLIDS

	Accuracy	Precision	Recall	F1 Score
Client 1	0.934479	0.934532	0.934479	0.934502
Client 2	0.935486	0.935577	0.935486	0.935524
Client 3	0.933790	0.933813	0.933790	0.933801
Server	0.997886	0.997888	0.997886	0.997887
Weighted Score	0.986729	0.986741	0.986729	0.986733
Majority Voting	0.971294	0.973226	0.971294	0.971448

their high effectiveness. The three client models consistently show performance metrics slightly above 0.93 across Accuracy, Precision, Recall, and F1 Score. This consistency indicates their robust capability in accurate prediction. The close alignment of all four metrics for each model suggests a well-balanced proficiency in identifying both positive and negative classes. The server model demonstrates nearly perfect scores in all metrics, exceeding 0.99, reflecting its superior predictive ability. This may be attributed to the more comprehensive training set used for the server model.

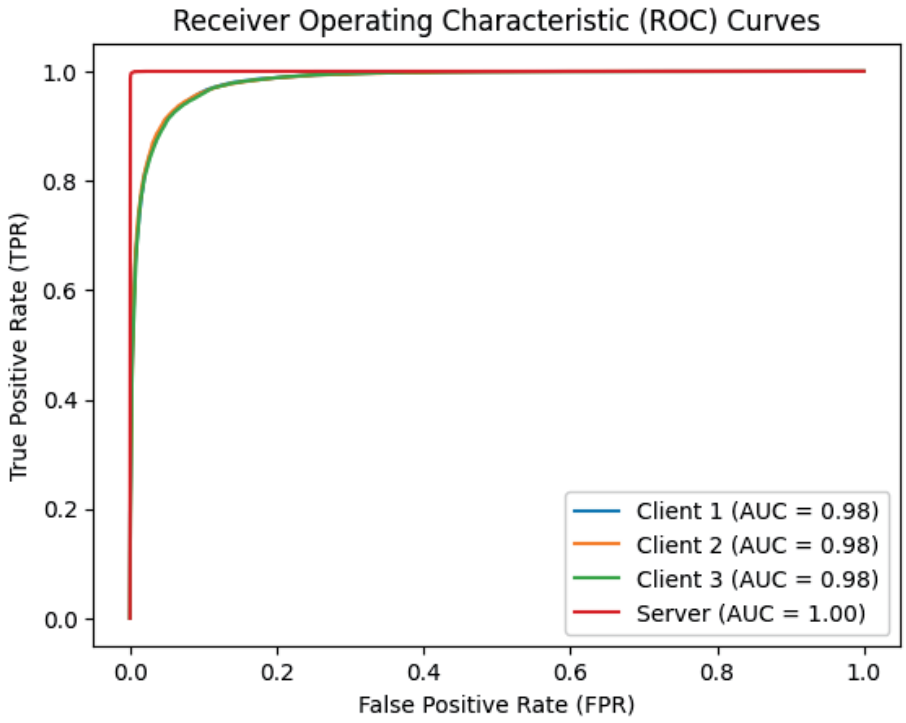

Figure 4.1: ROC Curves without Applying SC-MLIDS

The ROC curves, as presented in Figure 4.1, corroborate these results. All models give high area under the ROC curve (AUC) values, with the client models achieving

an AUC of 0.98 and the server model scoring 1.00. Such high AUC values represent remarkable model classification capabilities, further affirming the models' predictive strength.

However, the reliance on a single model type is not sufficient in the SC-MLIDS framework, which integrates both server and client models, each validating different data types. Hence, we implemented two model aggregation prediction algorithms to yield balanced and comprehensive prediction results.

Table 4.4 also presents the results of the weighted score algorithm, a scoring algorithm based on model performance and weights. The weights for the three client models and the server model were set at 0.2, 0.2, 0.2, and 0.4, respectively, reflecting the server model's higher performance and importance. The Precision weight was set at 0.6 and the F1 Score weight was set at 0.4. This specific weighting strategy was adopted because we emphasized minimizing false positives and achieving balanced predictions.

The weighted score algorithm combines the models to yield predictions that are tested on the testing set with all four metrics at approximately 0.9867. Similarly, the majority voting algorithm performs robustly, exceeding 0.97 in all metrics, with a slightly higher Precision. While both algorithms slightly lag behind the best-performing server model (by about 1% and 2%), they offer a more comprehensive intrusion detection capability, reflecting the integrated predictive power that individual models lack.

In the next phase of our experiment, we continued to evaluate algorithm performance by simulating the operational phase of the SC-MLIDS framework. Specifically, we simulated a sink node collecting data from its managed sensor nodes, validating it using the client model at the sink node, and then forwarding it to the gateway for aggregation prediction.

For this experiment, we used a balanced subset of the testing set, giving an example of Client 1, randomly selecting 10,000 samples each from positive and negative classes. Client 1 predicted 9,739 samples as negative and discarded them. Of the 10,261 samples predicted as positive and sent to the gateway, 9,451 were true positives, and 810 were false positives. Table 4.5 details the results of the aggregation prediction performed at the gateway.

Table 4.5: Simulation Results with SC-MLIDS Applied

	Accuracy	Precision	Recall	F1 Score
Client 1	0.921060	0.848352	0.921060	0.883212
Client 2	0.939869	0.940314	0.939869	0.940088
Client 3	0.942013	0.943665	0.942013	0.942789
Server	0.997759	0.997805	0.997759	0.997770
Weighted Score	0.997759	0.997805	0.997759	0.997770
Majority Voting	0.987623	0.989263	0.987623	0.988025

As expected, when data is validated by Client 1 with further filtering and validation, we observe normal Accuracy and Recall metrics of 0.92, with lower Precision and F1 Score. The other two client models gave slightly higher metrics, around 0.94. While close to previous results, the server model's performance showed a slight decrease, which may be caused by errors and remains within the range of normal fluctuations. Figure 4.2 illustrates the ROC curves for each model in the SC-MLIDS simulation, with respective AUC values of 0.92, 0.95, 0.95, and 1.00 for the three clients and the server.

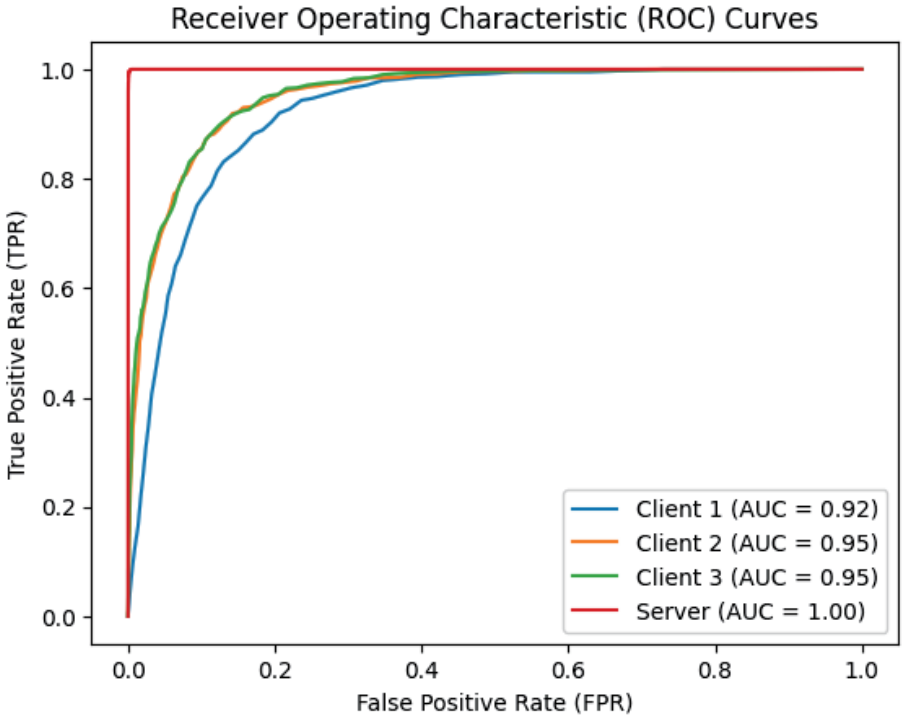

Figure 4.2: ROC Curves with SC-MLIDS Applied

However, the performance of the aggregation prediction algorithms gave significant

improvement when the SC-MLIDS framework was applied. Compared to direct testing, both algorithms showed improvements in all four metrics, by 1.12% and 1.68%, respectively. Notably, the weighted score algorithm's performance was identical to the best-performing server model, with all four metrics exceeding 0.99. The performance metrics of both the weighted score algorithm and the server model aligned for three reasons: firstly, the server model showed excellent performance; secondly, it assigned the greatest weight; thirdly, computational accuracy might introduce errors. Nevertheless, the weighted score algorithm results were calculated by aggregating the models, giving a significant difference from the server model's performance. In cases of exceptional model performance, the benefits of employing the weighted score algorithm are not readily apparent. However, in real WSN scenarios, the performance of individual models can vary significantly, leading to considerable differences in their prediction results. The weighted score algorithm can balance these results, overcoming the limitations of single models and offering comprehensive aggregated predictions.

Table 4.6: Simulation Results in Different Classifiers without SC-MLIDS

	Classifier	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Client 1	Random Forest	0.934331	0.934310	0.934331	0.934320
Client 2	Logistic Regression	0.617685	0.591091	0.617685	0.536262
Client 3	Gradient Boosting	0.733147	0.731853	0.733147	0.732405
Server	Support Vector Machine	0.970311	0.970298	0.970311	0.970303
Weighted Score	N/A	0.960628	0.961353	0.960628	0.960391
Majority Voting	N/A	0.942515	0.947397	0.942515	0.942962

In our further experiment with the proposed SC-MLIDS framework, we explored using various classifiers for generating models on the example of three clients. As detailed in Table 4.6, the classifiers deployed were RF for Client 1, LR for Client 2, and GB for Client 3, while the server model was trained using the SVM algorithm. Client 1 maintained the same performance metrics as the previous experiment due to the continued use of the RF classifier. The other two clients, however, showed poorer performance, with all metrics ranging between 0.53 and 0.73. In contrast, the server model demonstrated the highest performance, achieving an accuracy and other

metrics as high as 0.97.

When these models were merged using the two aggregation prediction algorithms, the synthesized prediction results for both algorithms were approximately 0.96 and 0.94 across all four metrics, respectively. Although these results were slightly lower than the best-performing server model, the aggregation achieved high accuracy in predictions, effectively harmonizing the individual models with varied performances.

Table 4.7: Simulation Results in Different Classifiers with SC-MLIDS

	Classifier	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Client 1	Random Forest	0.915598	0.838319	0.915598	0.875256
Client 2	Logistic Regression	0.859940	0.845874	0.859940	0.852753
Client 3	Gradient Boosting	0.773898	0.861784	0.773898	0.811472
Server	Support Vector Machine	0.975017	0.978432	0.975017	0.976061
Weighted Score	N/A	0.974149	0.977702	0.974149	0.975220
Majority Voting	N/A	0.958619	0.969096	0.958619	0.961657

Table 4.7 presents the results of the simulation runs for the SC-MLIDS framework using Client 1 as an example. After filtering the sensing data, the metrics showed varying degrees of improvement when predicted by each model. The most notable improvement was observed in Client 2, exceeding twenty percent. This improvement also positively impacted the performance of the model aggregation prediction algorithms, with both algorithms improving each metric by more than one percent compared to individual tests.

These experiments collectively demonstrate the performance of the SC-MLIDS framework and its aggregation prediction algorithms from multiple angles. When operating a WSN under the SC-MLIDS framework, data forwarded to the gateway experiences a two-layer validation process: first by the client model at the sink node and then by the synthesized verification at the gateway. This implementation of the aggregation prediction algorithms results in significantly enhanced accuracy of intrusion detection. Although the algorithm's performance metrics did not achieve a perfect model level, the comprehensive nature of the prediction approach employed ensures more reliable results.

Chapter 5

Discussion

5.1 Interpretation of Results

We conducted experiments in two parts, reflecting the two core components of the proposed Server-Client Machine Learning Intrusion Detection System (SC-MLIDS) framework: server-client communication simulation and aggregation prediction algorithms.

5.1.1 Server-Client Communication

Our simulation program, designed to demonstrate the operation of Wireless Sensor Networks (WSNs) applying the SC-MLIDS framework, includes two phases: the initial deployment phase and the formal operational phase.

During the initial deployment phase, the program demonstrated various processes such as client model training, communication and data transfer between the client and the server, server model training, and the execution of model aggregation prediction. The results of these experiments helped detail the consumption of computational and network resources, and the time spent in these activities. Notably, the SC-MLIDS framework, during its initial deployment phase, encountered potential additional resource consumption, primarily in the context of model training and transmission.

In our experiments, we consistently employed the Random Forest (RF) classifier, recognized for its high adaptability and performance in complex classification tasks. However, simple sensing data detection tasks may not require such complex Machine Learning (ML) algorithms. In such cases, employing simpler algorithms such as Decision Tree (DT) or Logistic Regression (LR) could lead to considerable savings in resources. This was evidenced in our subsequent experiments. With the same training set, a model trained using LR occupied only 2KB and required 0.2 seconds for training, significantly reducing resource consumption in the initial deployment phase.

In the formal operational phase, the client model was used for the initial detection and filtering of sensing data. Data identified as malicious by the client model was discarded, thus preventing the need for its transmission to higher levels, and consequently yielding benefits in terms of reduced resource consumption. This data filtering approach was specifically adopted to meet the unique requirements of WSN data, where minimizing false positives is more critical than reducing false negatives.

Furthermore, the model aggregation prediction at the gateways involved merging client and server models to collectively analyze and detect sensing and network traffic data. This task was easily manageable by the gateway equipped with superior hardware configurations. The deployment of various security measures in the gateway or base station is a well-established practice in WSNs, as corroborated by extensive research in the field.

The summarization and analysis of the experimental results from these phases indicated that the potential resource consumption resulting from the SC-MLIDS framework did not exceed the hardware and network resource limitations inherent in WSNs. This affirmed the framework's alignment with the need for lightweight solutions in WSNs.

5.1.2 Aggregation Prediction Algorithms

In the SC-MLIDS framework, the gateway plays a critical role by aggregating predictions from both client and server models. This process results in the generation of final detection results through the synthesis and analysis of filtered sensing data and network traffic data. This model aggregation is achieved through the implementation of two specially designed algorithms: weighted score and majority voting.

The weighted score algorithm is particularly designed for the concerns of false positives in WSNs. It uses a calculated combination of Precision and F1 Score, along with weights that are assigned based on the model's importance or the complexity of the task. This approach takes into account both the performance of the model and its assigned weight.

The majority voting algorithm relies on collective voting on the prediction results of the client models, a feature inherent in the structure of multiple client models and a single server model in SC-MLIDS. Since the server model is tasked with the

critical function of network traffic data detection, which is of higher complexity and importance, the results from the majority voting are matched with those of the server model to ensure more reliable predictions.

Although the foundational ideas of these algorithms are well-established in the field of ML, our adaptations are significantly customized to suit the specific characteristics of WSNs. Both algorithms are uniquely developed for model aggregation in the context of WSNs. Since both algorithms perform the same task of generating aggregated prediction results, the performance of both algorithms should be examined according to the WSN application scenarios and requirements when applying the SC-MLIDS framework, and one of them should be selected to perform the task.

In the context of intrusion detection, relying on the prediction results of a single model is often insufficient. Hence, we employ model aggregation prediction algorithms for a more comprehensive evaluation. Upon testing the models in the SC-MLIDS framework with a testing set, we found that the server model, benefiting from a more complete training set, showed superior performance. The client models, in contrast, showed slightly inferior performance compared to the server model.

Although the predictions generated by our two algorithms were marginally less effective than the optimally performing server model, the performance metrics of both algorithms still met the standards of excellence, with all metrics close to or exceeding 99%. Additionally, the reduced data transmission volume, a result of sensing data filtering by the client models, further enhanced the performance of both algorithms.

Notably, the high-accuracy detection capabilities of both algorithms are particularly remarkable in scenarios involving different types of classifiers or models with lower performance metrics. This adaptability and flexibility enable them to demonstrate exceptional detection capabilities, even under conditions of underperformance models or in the face of more complex tasks.

5.2 Implications

Current research on ML-based Intrusion Detection Systems (IDS) for WSNs primarily focuses on deploying specific ML algorithms to identify particular types of network attacks. However, the SC-MLIDS framework proposed in this thesis transcends these limitations. It is not restricted to detecting only certain types of attacks,

nor is it restricted in its selection of ML algorithms or the number of models in terms of its theoretical design. Depending on application requirements, various types of ML algorithms can be chosen, and models can be trained using comprehensive and complete datasets. This enables the detection of a wide range of attack types across different WSN architectures.

The SC-MLIDS framework segments intrusion detection into two phases, based on the data characteristics of WSNs. The initial detection is conducted at the sink node level using the client model, effectively filtering well-characterized malicious data and preventing unnecessary data transmission. Subsequently, at the gateway level, model aggregation prediction algorithms are employed to thoroughly validate both sensing data and network traffic data, achieving comprehensive intrusion detection.

The two model aggregation prediction algorithms integrated into the SC-MLIDS framework deliver comprehensive intrusion detection results. These results are derived by holistically considering the performance, weight, and task-specific characteristics of the models. Both algorithms are designed with the unique attributes of WSNs in mind, particularly focusing on the reliability of collected data and aiming to achieve superior intrusion detection with low false positive rates.

From a practical perspective, the proposed SC-MLIDS framework aligns with the typical three-layer architecture of WSNs: including sensor nodes, sink nodes, and gateway. Realizing two layers of intrusion detection at the sink node and gateway levels. This structural design offers the flexibility to adapt to WSN architectures of varying complexity, providing an innovative and comprehensive solution for WSN security. Moreover, our proposed framework efficiently uses the computational and network resources of each component within the WSN, meeting the network's requirement for a lightweight solution.

5.3 Limitations

The proposed framework has some limitations, each of which deserves consideration for future research.

The dataset used in our experiments may not accurately replicate real-world WSN datasets. The SC-MLIDS framework involves specific dataset characteristics, particularly containing both sensing data and corresponding network traffic data. Due to the

absence of suitable open-source datasets meeting these criteria, we used the TON_IoT dataset as an alternative. In our approach, sensing data served as the baseline, which we merged with network traffic data based on matching timestamps and labels. While this method is logically sound, it may introduce errors, potentially leading to minor biases in the models trained using this dataset.

There is a notable gap between our simulation programs and actual deployed applications. The simulation programs were designed to demonstrate the operational process of the SC-MLIDS framework, but the experimental data generated may differ from that in a real-world deployed WSN. Additionally, the process simulated in our program might not fully capture the complexities and nuances of an operational WSN, potentially leading to unforeseen issues in practical deployments.

Our experiments did not explore a wide range of combinations of ML algorithms. The primary focus was on using RF classifiers for model training. Although we also experimented with four different classifiers and observed varying results, the selection of ML algorithms has significant implications for the detection process and the overall performance of the SC-MLIDS. A more extensive exploration of algorithm combinations could potentially yield insights into optimizing the detection capabilities of our proposed framework.

Our proposed model aggregation prediction algorithms have shortcomings. Although our algorithms are flexible and adaptive to models with varying performance, no measures are taken for low-precision models that may have an impact on the prediction results. This may lead to a reduction in intrusion detection confidence. Additionally, the setting of model weights in the weighted score algorithm and the selection of performance metrics, along with their weight settings, can significantly impact the results of this algorithm. The choice of these parameters in this experiment may introduce some bias.

5.4 Future Work

To address the identified limitations, future enhancements to this thesis could be pursued in three key areas: dataset, algorithms, and deployment strategy.

For the enhancement of model accuracy and relevance, dataset selection should closely mirror actual application scenarios. Thus, deploying real WSNs to collect

real sensing data and network traffic data is imperative for advancing the SC-MLIDS framework. Additionally, launching various types of attacks on these networks would be beneficial to collect data containing a wider scope of attack scenarios, thereby strengthening the model's detection capabilities.

The selection of ML algorithms is critical to the effectiveness of intrusion detection models. As sensing data typically includes state information and parameters such as temperature, humidity, and pressure, such simple data could be used to train models using simple ML algorithms. This would enable client models to conduct initial intrusion detection with minimal resource consumption. Conversely, server models, tasked with the more complex job of detecting network traffic data, require more sophisticated ML algorithms due to the critical nature of this data for intrusion detection. Hence, extensive experimentation is necessary to identify suitable ML algorithms for both client and server models.

Improvements to model aggregation prediction algorithms are necessary. In model aggregation prediction, effective strategies include incorporating dynamic weight assignment based on model performance or excluding underperforming models and replacing them with superior-performing ones. Dynamic weight assignment automatically adjusts the weights based on the performance of each model in a particular situation, adapting to different data distributions and changes in the WSN environment. Additionally, promptly identifying and excluding underperforming models or replacing them with better-performing alternatives can enhance the holistic system performance. This strategy addresses the limitations of fixed weight settings in weighted score algorithms and enables the system to dynamically adapt to changing threats and data characteristics in real-time WSN environments.

The unique aspect of the SC-MLIDS framework is its integration of clients and a server, corresponding to the sink nodes and gateway in WSNs. In more straightforward WSNs, which might lack either sink nodes or gateway, the deployment of the SC-MLIDS framework could be constrained. In such cases, the functions of the client and server models could be merged within either the sink node or the gateway, enabling simultaneous detection of both data types. Conversely, in larger WSNs that include both sink nodes and a gateway, there is no barrier to deploying the SC-MLIDS.

Chapter 6

Conclusion

The Server-Client Machine Learning Intrusion Detection System (SC-MLIDS) proposed in this thesis introduces a novel hybrid Machine Learning (ML) framework for intrusion detection in Wireless Sensor Networks (WSNs). Drawing upon the concept of Federated Learning (FL), this framework takes the strengths of ML to minimize data transmission while ensuring high-precision intrusion detection, thereby effectively safeguarding WSN security.

In designing the SC-MLIDS framework, we have thoroughly considered the architectural specificities of WSNs and the necessity for data reliability. The framework effectively implements two-layer intrusion detection by deploying simplified client models at sink nodes and a more complex server model at the gateway. This framework strikes a balance between low resource consumption and enhanced detection accuracy. We have validated the effectiveness of this framework through a series of experiments, assessing both the individual performance of client and server models and the effectiveness of integrating the two proposed model aggregation prediction algorithms.

These algorithms, namely weighted score and majority voting, are designed to improve the reliability of prediction results. They are particularly designed to align with the specific characteristics of WSNs, such as data features and transmission constraints, thereby achieving efficient intrusion detection while maintaining a low false positive rate. Experimental results demonstrate that these algorithms excel in merging the prediction results of models with diverse performance levels. Specifically, their capacity to deliver comprehensive predictions, ensures that intrusion detection is not overly reliant on a single model.

Moreover, the design and implementation of SC-MLIDS underscore its adaptability for various types of attack detection and flexibility in selecting ML algorithms

based on actual application requirements. This flexibility makes it an effective breakthrough in WSN network security.

In conclusion, the SC-MLIDS framework demonstrates significant potential in the field of WSN intrusion detection, both theoretically and practically. It not only provides an efficient and high-precision detection methodology but also offers a new direction for future research in WSN security.

Bibliography

[1] A. Flammini, P. Ferrari, D. Marioli, E. Sisinni, and A. Taroni, "Wired and wireless sensor networks for industrial applications," *Microelectronics journal*, vol. 40, no. 9, pp. 1322–1336, 2009.

[2] M. N. Mowla, N. Mowla, A. S. Shah, K. Rabie, and T. Shongwe, "Internet of things and wireless sensor networks for smart agriculture applications-a survey," *IEEE Access*, 2023.

[3] O. Can and O. K. Sahingoz, "A survey of intrusion detection systems in wireless sensor networks," in *2015 6th international conference on modeling, simulation, and applied optimization (ICMSAO)*, pp. 1–6, IEEE, 2015.

[4] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli, "A detailed investigation and analysis of using machine learning techniques for intrusion detection," *IEEE communications surveys & tutorials*, vol. 21, no. 1, pp. 686–728, 2018.

[5] F. Gaojuan, W. Ruchuan, H. Haiping, S. Lijuan, and X. Fu, "Performance analysis for intrusion target detection in wireless sensor networks," *Chinese Journal of Electronics*, vol. 20, no. 4, pp. 725–729, 2011.

[6] W. Wang, H. Huang, Q. Li, F. He, and C. Sha, "Generalized intrusion detection mechanism for empowered intruders in wireless sensor networks," *IEEE Access*, vol. 8, pp. 25170–25183, 2020.

[7] D.-f. Ye, L.-l. Min, and W. Wang, "Design and implementation of wireless sensor network gateway based on environmental monitoring," in *2009 International Conference on Environmental Science and Information Application Technology*, vol. 2, pp. 289–292, IEEE, 2009.

[8] M. Kocakulak and I. Butun, "An overview of wireless sensor networks towards internet of things," in *2017 IEEE 7th annual computing and communication workshop and conference (CCWC)*, pp. 1–6, Ieee, 2017.

[9] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer networks*, vol. 52, no. 12, pp. 2292–2330, 2008.

[10] K. Bagadi, R. Cv, and K. Sathish, "An overview of localization techniques in underwater wireless sensor networks," in *2022 Third International Conference on Intelligent Computing Instrumentation and Control Technologies (ICICICT)*, pp. 1687–1692, IEEE, 2022.

[11] H. Guo, K.-S. Low, and H.-A. Nguyen, "Optimizing the localization of a wireless sensor network in real time based on a low-cost microcontroller," *IEEE transactions on industrial electronics*, vol. 58, no. 3, pp. 741–749, 2009.

[12] A. Khalifeh, F. Mazunga, A. Nechibvute, and B. M. Nyambo, "Microcontroller unit-based wireless sensor network nodes: A review," *Sensors*, vol. 22, no. 22, p. 8937, 2022.

[13] Y. Gu, F. Ren, Y. Ji, and J. Li, "The evolution of sink mobility management in wireless sensor networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 507–524, 2015.

[14] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Communications magazine*, vol. 40, no. 8, pp. 102–114, 2002.

[15] A. Ali, Y. Ming, S. Chakraborty, and S. Iram, "A comprehensive survey on real-time applications of wsn," *Future internet*, vol. 9, no. 4, p. 77, 2017.

[16] C. Buratti, A. Conti, D. Dardari, and R. Verdone, "An overview on wireless sensor networks technology and evolution," *Sensors*, vol. 9, no. 9, pp. 6869–6896, 2009.

[17] L. d. T. Steenkamp, S. Kaplan, and R. H. Wilkinson, "Wireless sensor network gateway," in *AFRICON 2009*, pp. 1–6, IEEE, 2009.

[18] J. Kim and D. Choi, "esgate: Secure embedded gateway system for a wireless sensor network," in *2008 IEEE International Symposium on Consumer Electronics*, pp. 1–4, IEEE, 2008.

[19] M. A. Hussain, K. kyung Sup, *et al.*, "Wsn research activities for military application," in *2009 11th international conference on advanced communication technology*, vol. 1, pp. 271–274, IEEE, 2009.

[20] P.-C. Hii and W.-Y. Chung, "A comprehensive ubiquitous healthcare solution on an android™ mobile device," *Sensors*, vol. 11, no. 7, pp. 6799–6815, 2011.

[21] D. D. K. Rathinam, D. Surendran, A. Shilpa, A. S. Grace, and J. Sherin, "Modern agriculture using wireless sensor network (wsn)," in *2019 5th international conference on advanced computing & communication Systems (ICACCS)*, pp. 515–519, IEEE, 2019.

[22] R. S. Ransing and M. Rajput, "Smart home for elderly care, based on wireless sensor network," in *2015 International Conference on Nascent Technologies in the Engineering Field (ICNTE)*, pp. 1–5, IEEE, 2015.

[23] S. Mansour, N. Nasser, L. Karim, and A. Ali, "Wireless sensor network-based air quality monitoring system," in *2014 international conference on computing, networking and communications (ICNC)*, pp. 545–550, IEEE, 2014.

[24] J. Zhang, W. Li, N. Han, and J. Kan, "Forest fire detection system based on a zigbee wireless sensor network," *Frontiers of Forestry in China*, vol. 3, pp. 369–374, 2008.

[25] J. Ben-Othman and B. Yahya, "Energy efficient and qos based routing protocol for wireless sensor networks," *Journal of Parallel and Distributed Computing*, vol. 70, no. 8, pp. 849–857, 2010.

[26] M. A. M. Vieira, C. N. Coelho, D. j. da Silva, and J. M. da Mata, "Survey on wireless sensor network devices," in *EFTA 2003. 2003 IEEE Conference on Emerging Technologies and Factory Automation. Proceedings (Cat. No. 03TH8696)*, vol. 1, pp. 537–544, IEEE, 2003.

[27] M. N. U. Islam, A. Fahmin, M. S. Hossain, and M. Atiquzzaman, "Denial-of-service attacks on wireless sensor network and defense techniques," *Wireless Personal Communications*, vol. 116, pp. 1993–2021, 2021.

[28] A.-S. K. Pathan, H.-W. Lee, and C. S. Hong, "Security in wireless sensor networks: issues and challenges," in *2006 8th International Conference Advanced Communication Technology*, vol. 2, pp. 6–pp, IEEE, 2006.

[29] G. A. N. Segura, S. Skaperas, A. Chorti, L. Mamatas, and C. B. Margi, "Denial of service attacks detection in software-defined wireless sensor networks," in *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 1–7, IEEE, 2020.

[30] M. Jamshidi, E. Zangeneh, M. Esnaashari, A. M. Darwesh, and M. R. Meybodi, "A novel model of sybil attack in cluster-based wireless sensor networks and propose a distributed algorithm to defend it," *Wireless Personal Communications*, vol. 105, pp. 145–173, 2019.

[31] J. Wadii, H. Rim, and B. Ridha, "Detecting and preventing sybil attacks in wireless sensor networks," in *2019 IEEE 19th Mediterranean Microwave Symposium (MMS)*, pp. 1–5, IEEE, 2019.

[32] S. Ali, M. A. Khan, J. Ahmad, A. W. Malik, and A. ur Rehman, "Detection and prevention of black hole attacks in iot & wsn," in *2018 third international conference on fog and mobile edge computing (FMEC)*, pp. 217–226, IEEE, 2018.

[33] B. K. Mishra, M. C. Nikam, and P. Lakkadwala, "Security against black hole attack in wireless sensor network-a review," in *2014 Fourth International Conference on Communication Systems and Network Technologies*, pp. 615–620, IEEE, 2014.

[34] M. Wazid, A. Katal, R. S. Sachan, R. Goudar, and D. Singh, "Detection and prevention mechanism for blackhole attack in wireless sensor network," in *2013 International Conference on Communication and Signal Processing*, pp. 576–581, IEEE, 2013.

[35] A. Tayebi, S. Berber, and A. Swain, "Wireless sensor network attacks: An overview and critical analysis," in *2013 Seventh international conference on sensing technology (ICST)*, pp. 97–102, IEEE, 2013.

[36] R. K. Dwivedi, P. Sharma, and R. Kumar, "Detection and prevention analysis of wormhole attack in wireless sensor network," in *2018 8th international conference on cloud computing, data science & engineering (confluence)*, pp. 727–732, IEEE, 2018.

[37] G. Farjamnia, Y. Gasimov, and C. Kazimov, "Review of the techniques against the wormhole attacks on wireless sensor networks," *Wireless Personal Communications*, vol. 105, pp. 1561–1584, 2019.

[38] P. Baronti, P. Pillai, V. W. Chook, S. Chessa, A. Gotta, and Y. F. Hu, "Wireless sensor networks: A survey on the state of the art and the 802.15. 4 and zigbee standards," *Computer communications*, vol. 30, no. 7, pp. 1655–1695, 2007.

[39] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: Attacks and countermeasures," *Ad hoc networks*, vol. 1, no. 2-3, pp. 293–315, 2003.

[40] S. Banga, H. Arora, S. Sankhla, G. Sharma, and B. Jain, "Performance analysis of hello flood attack in wsn," in *Proceedings of International Conference on Communication and Computational Technologies: ICCCT-2019*, pp. 335–342, Springer, 2020.

[41] K. Saghar, D. Kendall, and A. Bouridane, "Raeed: A solution for hello flood attack," in *2015 12th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pp. 248–253, IEEE, 2015.

[42] L. Da Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on industrial informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[43] S. Li, L. D. Xu, and S. Zhao, "The internet of things: a survey," *Information systems frontiers*, vol. 17, pp. 243–259, 2015.

[44] K. M. Sadique, R. Rahmani, and P. Johannesson, "Towards security on internet of things: applications and challenges in technology," *Procedia Computer Science*, vol. 141, pp. 199–206, 2018.

[45] L. Tan and N. Wang, "Future internet: The internet of things," in *2010 3rd international conference on advanced computer theory and engineering (ICACTE)*, vol. 5, pp. V5–376, IEEE, 2010.

[46] B. B. Gupta and M. Quamara, "An overview of internet of things (iot): Architectural aspects, challenges, and protocols," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 21, p. e4946, 2020.

[47] P. Asghari, A. M. Rahmani, and H. H. S. Javadi, "Internet of things applications: A systematic review," *Computer Networks*, vol. 148, pp. 241–261, 2019.

[48] X. Li, R. Lu, X. Liang, X. Shen, J. Chen, and X. Lin, "Smart community: an internet of things application," *IEEE Communications magazine*, vol. 49, no. 11, pp. 68–75, 2011.

[49] Y. Yuehong, Y. Zeng, X. Chen, and Y. Fan, "The internet of things in healthcare: An overview," *Journal of Industrial Information Integration*, vol. 1, pp. 3–13, 2016.

[50] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, "A review of applications in federated learning," *Computers & Industrial Engineering*, vol. 149, p. 106854, 2020.

[51] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, "Federated learning for internet of things: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1622–1658, 2021.

[52] S. Banabilah, M. Aloqaily, E. Alsayed, N. Malik, and Y. Jararweh, "Federated learning review: Fundamentals, enabling technologies, and future applications," *Information processing & management*, vol. 59, no. 6, p. 103061, 2022.

[53] L. U. Khan, W. Saad, Z. Han, E. Hossain, and C. S. Hong, "Federated learning for internet of things: Recent advances, taxonomy, and open challenges," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1759–1799, 2021.

[54] J. Liu, J. Huang, Y. Zhou, X. Li, S. Ji, H. Xiong, and D. Dou, "From distributed machine learning to federated learning: A survey," *Knowledge and Information Systems*, vol. 64, no. 4, pp. 885–917, 2022.

[55] S. Niknam, H. S. Dhillon, and J. H. Reed, "Federated learning for wireless communications: Motivation, opportunities, and challenges," *IEEE Communications Magazine*, vol. 58, no. 6, pp. 46–51, 2020.

[56] T. N. Rincy and R. Gupta, "Ensemble learning techniques and its efficiency in machine learning: A survey," in *2nd international conference on data, engineering and applications (IDEA)*, pp. 1–6, IEEE, 2020.

[57] I. D. Mienye and Y. Sun, "A survey of ensemble learning: Concepts, algorithms, applications, and prospects," *IEEE Access*, vol. 10, pp. 99129–99149, 2022.

[58] K. A. Nguyen, W. Chen, B.-S. Lin, and U. Seeboonruang, "Comparison of ensemble machine learning methods for soil erosion pin measurements," *ISPRS International Journal of Geo-Information*, vol. 10, no. 1, p. 42, 2021.

[59] G. Ngo, R. Beard, and R. Chandra, "Evolutionary bagging for ensemble learning," *Neurocomputing*, vol. 510, pp. 1–14, 2022.

[60] F. Huang, G. Xie, and R. Xiao, "Research on ensemble learning," in *2009 International Conference on Artificial Intelligence and Computational Intelligence*, vol. 3, pp. 249–252, IEEE, 2009.

[61] R. Jurdak, X. R. Wang, O. Obst, and P. Valencia, "Wireless sensor network anomalies: Diagnosis and detection strategies," in *Intelligence-Based Systems Engineering*, pp. 309–325, Springer, 2011.

[62] M. A. Alsheikh, S. Lin, D. Niyato, and H.-P. Tan, "Machine learning in wireless sensor networks: Algorithms, strategies, and applications," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1996–2018, 2014.

[63] M. Mamdouh, M. A. Elrukhsi, and A. Khattab, "Securing the internet of things and wireless sensor networks via machine learning: A survey," in *2018 International Conference on Computer and Applications (ICCA)*, pp. 215–218, IEEE, 2018.

[64] S. Ismail, T. T. Khoei, R. Marsh, and N. Kaabouch, "A comparative study of machine learning models for cyber-attacks detection in wireless sensor networks," in *2021 IEEE 12th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pp. 0313–0318, IEEE, 2021.

[65] V. Gowdhaman and R. Dhanapal, "An intrusion detection system for wireless sensor networks using deep neural network," *Soft Computing*, pp. 1–9, 2021.

[66] S. Otoum, B. Kantarci, and H. T. Mouftah, "A novel ensemble method for advanced intrusion detection in wireless sensor networks," in *Icc 2020-2020 ieee international conference on communications (icc)*, pp. 1–6, IEEE, 2020.

[67] S. Umamaheshwari, S. A. Kumar, and S. Sasikala, "Towards building robust intrusion detection system in wireless sensor networks using machine learning and feature selection," in *2021 international conference on advancements in electrical, electronics, communication, computing and automation (ICAECA)*, pp. 1–6, IEEE, 2021.

[68] R. Ul Islam, M. S. Hossain, and K. Andersson, "A novel anomaly detection algorithm for sensor data under uncertainty," *Soft Computing*, vol. 22, no. 5, pp. 1623–1639, 2018.

[69] P. Gulganwa and S. Jain, "Ees-wca: energy efficient and secure weighted clustering for wsn using machine learning approach," *International Journal of Information Technology*, vol. 14, no. 1, pp. 135–144, 2022.

[70] Z. Yu and J. J. Tsai, "A framework of machine learning based intrusion detection for wireless sensor networks," in *2008 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (sutc 2008)*, pp. 272–279, IEEE, 2008.

[71] K. Medhat, R. A. Ramadan, and I. Talkhan, "Distributed intrusion detection system for wireless sensor networks," in *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies*, pp. 234–239, IEEE, 2015.

[72] W. Zhang, D. Han, K.-C. Li, and F. I. Massetto, "Wireless sensor network intrusion detection system based on mk-elm," *Soft Computing*, vol. 24, pp. 12361–12374, 2020.

[73] Y. Maleh, A. Ezzati, Y. Qasmaoui, and M. Mbida, "A global hybrid intrusion detection system for wireless sensor networks," *Procedia Computer Science*, vol. 52, pp. 1047–1052, 2015.

[74] N. M. Alruhaily and D. M. Ibrahim, "A multi-layer machine learning-based intrusion detection system for wireless sensor networks," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 4, 2021.

[75] T. M. Booij, I. Chiscop, E. Meeuwissen, N. Moustafa, and F. T. Den Hartog, "Ton_iot: The role of heterogeneity and the need for standardization of features and attack types in iot network intrusion data sets," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 485–496, 2021.

[76] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[77] A. More and D. P. Rana, "Review of random forest classification techniques to resolve data imbalance," in *2017 1st International Conference on Intelligent Systems and Information Management (ICISIM)*, pp. 72–78, IEEE, 2017.

[78] J. L. Speiser, M. E. Miller, J. Tooze, and E. Ip, "A comparison of random forest variable selection methods for classification prediction modeling," *Expert systems with applications*, vol. 134, pp. 93–101, 2019.

Appendix A

Server-Client Communication

A.1 Server Console Result

```
C:\Users>python helper.py
C:\Users>python server.py
Server is waiting for connections ...

Connected to Client 1:
Address: ('127.0.0.1', 54742)
Client 1 model decryption information:
Key: b'ZJg3BjY6SI0hL9h8IYYf9GxsUR5aqtd6ntQU5wXpjTs='
Seed: 0.60792627
Salt: b'\x03\x02\x8a\xcd\xe5\xd2\xdf\xfa\x00q\xc3\xee~\x85\xef\xdf'
Receiving Client 1 model file: 100%|108M/108M [01:20<00:00, 1.41MB/s]
Client 1 model file has been received, time spent: 80.2155 seconds
Client 1 model file size: 107.5145 MB
Client 1 model file has been decompressed and decrypted
    , time spent: 1.1003 seconds
Client 1 model file has been saved to: ./received_models/client_1.joblib
------------------------ Client 1 Completed ------------------------

Connected to Client 2:
Address: ('127.0.0.1', 54814)
Client 2 model decryption information:
Key: b'D39wlizQSuynNVIslW_DzbkqkvO__AjdaQt5x_A9jmQ='
Seed: 0.60615668
Salt: b'\xca\xb9\x92\x11>\xe1\x97\xc1\xf6\x13I\x04NQ{"'
Receiving Client 2 model file: 100%|108M/108M [01:04<00:00, 1.74MB/s]
```

```
Client 2 model file has been received, time spent: 64.8709 seconds
Client 2 model file size: 107.7409 MB
Client 2 model file has been decompressed and decrypted
    , time spent: 0.8955 seconds
Client 2 model file has been saved to: ./received_models/client_2.joblib
------------------------ Client 2 Completed ------------------------

Connected to Client 3:
Address: ('127.0.0.1', 54830)
Client 3 model decryption information:
Key: b'yXP5THp4LPQSTGB_-xy8_x0H8HzaGJoUlzHZ6KfVWoY='
Seed: 0.60702304
Salt: b'\xbc\xd9xMw0\x87d\x91\x9e\xb5\xa8\xb9\xa8z\xd6'
Receiving Client 3 model file: 100%|107M/107M [01:16<00:00, 1.48MB/s]
Client 3 model file has been received, time spent: 76.1133 seconds
Client 3 model file size: 107.3119 MB
Client 3 model file has been decompressed and decrypted
    , time spent: 1.1386 seconds
Client 3 model file has been saved to: ./received_models/client_3.joblib
------------------------ Client 3 Completed ------------------------

----------------- All Clients Have Been Processed -----------------

Training the server model using network traffic data ...
Target distribution: {0: 98809, 1: 63943}
Server model training completed in 6.8305 seconds.
Server model saved to: ./received_models/server_model.joblib
---------------------------- All Done ----------------------------
```

A.2 Clients Console Result

```
C:\Users>python client1.py
Client 1:
```

```
Target distribution: {0: 32980, 1: 21270}
Client 1 model training completed in 8.9069 seconds.
Client 1 model saved to: ./client_models/client_1.joblib
Client 1 model encryption information:
Key: b'ZJg3BjY6SI0hL9h8IYYf9GxsUR5aqtd6ntQU5wXpjTs='
Seed: 0.60792627
Salt: b'\x03\x02\x8a\xcd\xe5\xd2\xdf\xfa\x00q\xc3\xee~\x85\xef\xdf'
Model file size: 106.4457 MB
Model file encrypted, file size: 141.9277 MB
    , time spent: 0.7266 seconds
Model file compressed, file size: 107.5145 MB
    , time spent: 5.2522 seconds
Client 1 has connected to the server
Model file has been sent from Client 1, file size: 107.5145 MB
    , time spent: 0.0100 seconds
----------------------- Client 1 Completed -----------------------

C:\Users>python client2.py
Client 2:
Target distribution: {0: 32884, 1: 21366}
Client 2 model training completed in 8.3103 seconds.
Client 2 model saved to: ./client_models/client_2.joblib
Client 2 model encryption information:
Key: b'D39wlizQSuynNVIslW_DzbkqkvO__AjdaQt5x_A9jmQ='
Seed: 0.60615668
Salt: b'\xca\xb9\x92\x11>\xe1\x97\xc1\xf6\x13I\x04NQ{"'
Model file size: 106.6699 MB
Model file encrypted, file size: 142.2266 MB
    , time spent: 0.5947 seconds
Model file compressed, file size: 107.7409 MB
    , time spent: 3.8984 seconds
Client 2 has connected to the server
```

```
Model file has been sent from Client 2, file size: 107.7409 MB
    , time spent: 0.0080 seconds
------------------------ Client 2 Completed ------------------------

C:\Users>python client3.py
Client 3:
Target distribution: {0: 32931, 1: 21319}
Client 3 model training completed in 8.1050 seconds.
Client 3 model saved to: ./client_models/client_3.joblib
Client 3 model encryption information:
Key: b'yXP5THp4LPQSTGB_-xy8_x0H8HzaGJoUlzHZ6KfVWoY='
Seed: 0.60702304
Salt: b'\xbc\xd9xMwO\x87d\x91\x9e\xb5\xa8\xb9\xa8z\xd6'
Model file size: 106.2449 MB
Model file encrypted, file size: 141.6600 MB
    , time spent: 0.5654 seconds
Model file compressed, file size: 107.3119 MB
    , time spent: 3.8393 seconds
Client 3 has connected to the server
Model file has been sent from Client 3, file size: 107.3119 MB
    , time spent: 0.0080 seconds
------------------------ Client 3 Completed ------------------------
```