UNIVERSITÄT BONN

Resilient controller against cyber threats in software-defined IoT networks

Master Thesis

Olid Mohd Sayed

Matriculation Number

3161945

This work was submitted to the

Institute of Computer Science IV

University of Bonn, Germany

Adviser(s):

Dr. Paulo Henrique Lopes Rettore
Dr. Bruno P. Santos
Mr. Sean Kloth

Examiners:

Prof. Dr. Michael Meier and Dr. Paulo Henrique Lopes Rettore

Registration date: 18-07-2025
Submission date: 19-01-2026

In collaboration with the Fraunhofer Institute for Communication, Information Processing and Ergonomics (FKIE), Bonn, Germany

UNIVERSITÄT BONN

Rheinische
Friedrich-Wilhelms-
Universität Bonn

Prüfungsausschuss Informatik

universität **bonn** · Institut für Informatik · 53012 Bonn

Vorsitzende des Prüfungsausschusses Prof. Dr. Anne Driemel

Stellvertretender Vorsitzender Prof. Dr. Thomas Kesselheim

Prüfungsamt:
Judith König
Tel.: 0228/73-4418
pa@informatik.uni-bonn.de
Postanschrift
Friedrich-Hirzebruch-Allee 5
Besucheranschrift:
Friedrich-Hirzebruch-Allee 8
53115 Bonn

www.informatik.uni-bonn.de

Erklärung über das selbständige Verfassen einer Abschlussarbeit
Declaration of Authorship

Titel der Arbeit/Title:

Resilient controller against cyber threats in software-defined IoT networks

Hiermit versichere ich / I hereby certify Sayed, Olid Mohd
Name/name Vorname / first name

dass ich die oben genannte Arbeit – bei einer Gruppenarbeit meinen entsprechend gekennzeichneten Anteil der Arbeit – selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

that I have written the above-mentioned work – in the case of group work, my correspondingly marked part of the work – independently and that I have not used any sources or resources other than those indicated and that I have identified all quotations.

Bonn, 19.01.2026

Olid

Unterschrift (im Original einzureichen im Prüfungsamt Informatik)

Abstract

Software-Defined Networking (SDN) is increasingly adopted in Internet of Things (IoT) environments to provide flexible and centralized control. However, securing SDN-based IoT networks remains challenging due to device heterogeneity, distributed operation, and strict resource constraints. In particular, centralized intrusion detection mechanisms lack visibility into host-level resource stress, while purely local approaches are limited by constrained computation and data availability. This thesis addresses these challenges by proposing a Federated Cyber Defense Agent (FCDA) to support cyber threat detection in SDN-based IoT environments. The FCDA is designed as a distributed anomaly detection component that complements an existing Cyber Defense Agent (CDA) and integrates network traffic metrics with host-level hardware resource monitoring, using a two-phase design comprising offline knowledge generation and online Federated Learning (FL). Offline training produces deployment artifacts, including anomaly detection models, normalization parameters, and detection thresholds, enabling immediate detection upon deployment, while online operation allows clients to perform local detection and federated model updates without sharing raw data. The proposed framework is implemented and evaluated under resource-constrained conditions, where experimental results demonstrate stable detection performance with high accuracy and precision, low false-positive rates, and minimal runtime overhead. Additional experiments show that the framework scales effectively from small to larger federations, maintaining consistent detection performance as the number of clients increases.

Acknowledgments

To my supervisors, Dr. Paulo Henrique Lopes Rettore, Dr. Bruno P. Santos, and Mr. Sean Kloth, I am deeply grateful for your guidance, mentorship, and constructive feedback throughout my thesis work. Your encouragement and confidence in my abilities were invaluable to the successful completion of this research. To my family and friends, thank you for your continued encouragement, patience, and support throughout this journey.

Contents

1

Introduction

As the IoT continues to expand, integrating billions of devices into interconnected systems, SDN has emerged as a vital architecture to support this growth [3]. SDN enables centralized control and dynamic programmability of networks, providing significant advantages in managing the complexity and scalability of IoT ecosystems. By decoupling the control and data planes, SDN facilitates seamless network configuration and optimization, making it an ideal choice for resource-constrained IoT environments [3]. However, this shift toward SDN-based IoT networks also introduces critical cybersecurity challenges that threaten the stability and security of these systems.

The SDN controller, often referred to as the brain of the network, is particularly vulnerable to cyber threats. Attacks such as Distributed Denial-of-Service (DDoS), flow table overflow, and malicious packet injections can disrupt network functionality, compromise data integrity, and even lead to controller failure [8]. These risks are worsened in IoT environments, where the vast number of devices and their diverse communication protocols increase the attack surface. Furthermore, the resource constraints of IoT devices, including limited CPU, memory, and bandwidth, hinder the effectiveness of traditional detection mechanisms that primarily rely on analyzing network traffic patterns and packet headers.

Current research has made significant progress in developing mitigation strategies for SDN cyber threats, such as advanced traffic analysis algorithms and flow table management techniques [6, 15]. However, these methods often operate in isolation, failing to leverage the potential of hardware-level insights. Resource anomalies, such as unusual CPU and memory usage patterns, remain an underexplored dimension in threat detection. These anomalies can serve as early indicators of malicious activities, particularly in IoT setups where conventional traffic-based methods struggle due to high resource contention and limited scalability.

Recognizing this gap, this thesis extends the foundational work of [12, 13] by introducing a resilient security framework for SDN-based IoT networks that leverages distributed learning. This approach uniquely integrates hardware monitoring,

specifically CPU and memory usage, along with traditional network analysis. By combining these diverse data sources within a FL paradigm, the proposed solution provides a dynamic, adaptive mechanism for detecting and mitigating cyber threats in real time, offering a comprehensive understanding of the network's hardware and software behavior.

This work presents a new paradigm in SDN security for IoT networks, addressing the dual challenges of scalability and vulnerability in resource-constrained environments. The findings contribute to more robust and adaptive security solutions, ensuring the safe and efficient operation of IoT ecosystems.

1.1 Problem Statement

The growing integration of IoT devices within SDN frameworks opens up new possibilities for enhanced connectivity and centralized network management. However, this expansion also increases the risk of cyber-security threats, which can significantly undermine the stability and integrity of IoT networks. Most existing research, including prior studies such as those presented in [12] and [13], has focused on detecting and mitigating cyber attacks primarily through network-based analytical methods, primarily by monitoring network traffic and analyzing packet headers. While these approaches have shown effectiveness, they often fail to address a critical limitation present in IoT environments: resource constraints. IoT devices typically operate with limited CPU, memory, and bandwidth, creating a unique challenge for implementing effective, sustainable security solutions.

Conventional threat detection techniques are tailored for more powerful computing environments, where computational resources are abundant. These methods often depend heavily on comprehensive, ongoing network traffic analysis, a process that can be resource-intensive and difficult to maintain in the resource-limited settings of IoT. The dilemma is that, although traffic-based techniques are useful for identifying suspicious behaviors, they can exhaust the limited CPU and memory resources of IoT devices, resulting in reduced performance and an increased risk of system failure. Therefore, these methods may not efficiently detect threats in SDN-IoT networks, where resource limitations hinder the accuracy and speed required for real-time threat identification.

Another major issue is the reliance on network traffic data alone, which may not always reveal the full extent of potential threats. Attacks that subtly manipulate resource utilization, such as spikes in CPU and memory usage, might remain unnoticed if only network traffic is evaluated. These subtle hardware performance changes could serve as early warning signs of an imminent cyberattack. However, few studies have investigated how integrating hardware monitoring with network analysis could improve threat detection. By neglecting hardware resource metrics, existing solutions overlook the opportunity to identify threats based on anomalies in CPU and memory usage, which could be particularly significant in IoT devices, where even slight variations may indicate malicious actions.

This gap in current research underscores the need for a more robust SDN controller that can address both network- and hardware-based threats in IoT environments.

The challenge lies in creating an adaptive, lightweight security mechanism that can adjust its threat-detection parameters in real time based on resource usage without overloading the system's limited capabilities. Furthermore, reliance on centralized analysis may significantly intensify communication overhead in large-scale IoT deployments. A distributed learning-based solution, such as FL, offers a promising approach to mitigating attacks by reducing the need to transmit raw data to a central controller, thereby lowering communication overhead and better accommodating resource-constrained devices. The development of such a solution requires novel techniques to integrate CPU and memory usage metrics with network data, enabling a more comprehensive and in-depth understanding of network security. Addressing these challenges is essential to establishing a resilient security framework for SDN-driven IoT networks, ensuring these systems can continue to function securely despite the inherent constraints of IoT environments.

1.2 Solution

This thesis addresses the challenge of providing secure SDN based IoT, with an emphasis on providing a FCDA for the use in distributed and resource-limited SDN-IoT systems. The agent will utilize both hardware system-level monitoring at the host level and network-level monitoring, utilizing a federated machine learning approach to provide an adaptive, privacy-preserving method to detect anomalies on heterogeneous devices in the IoT.

The FCDA is a decentralized detection framework comprising two interdependent operating phases: the Offline Knowledge Generation Phase and the Online Monitoring and Federated Adaptation Phase. In the offline phase, the FCDA uses historical data collected from IoT hosts to train an unsupervised anomaly detection model via hyperparameter tuning and federated training. Once an anomaly detection model has been generated during the offline phase, the FCDA generates the deployment artifacts needed to consistently and efficiently initialize the detection components in the online phase.

In the online stage, lightweight client agents will be deployed on each IoT host to collect system-level metrics (e.g., CPU and Memory usage) and network-level traffic characteristics. Each client agent will perform real-time anomaly detection using a reconstruction-based autoencoder and participate in FL to update the global model over time. A centralized federated server will coordinate the exchange of models between clients using a momentum-based aggregation strategy, allowing the system to adapt to evolving workloads and attack patterns while maintaining data locality and minimizing communication overhead.

The FCDA will focus on distributed anomaly detection and adaptive learning in order to identify potential threats. The FCDA will generate alerts when it detects anomalous activity and these alerts will be sent to the CDA[12, 13] and the CDA will take action to mitigate the threat. In this way, the proposed agent provides an additional layer of protection to the current agents by sending alert information to the CDA and allowing the CDA to perform various countermeasures such as modifying flow rules, isolating switches, reconfiguring controllers, and eliminates duplication of mitigation logic. This design separates responsibility of detection

and response, maintains compatibility with previous work and improves the overall system's ability to withstand attacks through the provision of timely and accurate detection information to the CDA.

1.3 Thesis Structure

This thesis is structured as follows. After this introduction, Chapter 2 presents the foundational concepts of Software-Defined Networking and IoT security, followed by a comprehensive review of existing research on cyber threat detection and mitigation in SDN-based IoT environments. Chapter 3 describes the design and architecture of the proposed FCDA, detailing both offline knowledge generation and online monitoring and adaptation mechanisms. In Chapter 4, the proposed design is evaluated in a MininetFed-based environment to assess its effectiveness, advantages, and limitations. Finally, Chapter 5 concludes the thesis by summarizing the contributions and outlining directions for future work.

2

Background

SDN has emerged as a transformative architecture for managing network infrastructure, particularly in IoT environments. By decoupling the control plane from the data plane, SDN enables centralized network management through a programmable controller, offering flexibility and dynamic configuration capabilities essential for IoT deployments. However, it also raises new security concerns beyond those inherent in SDN.

The central controller in SDN manages the flow tables, routing information, and network policies. However, when operating in IoT environments, the controller must do so while constrained by very limited CPU, memory, and power resources. The most common protocol for communication between controllers and switches in SDN architectures is OpenFlow. Flow tables in SDN switches maintain entries that dictate packet forwarding behavior, each specifying match conditions and corresponding actions.

Cyber threats against SDN-based IoT networks target vulnerabilities across the network's layers. A DDoS attack can flood the controller with traffic and exhaust its computational resources, rendering the network unable to continue functioning. Additionally, flow table overflow attacks can flood a switch with many flow entries, consuming all available space and limiting the switch's ability to forward packets, thereby degrading network performance. Lastly, Man in the Middle (MitM) attacks can compromise the integrity of data by intercepting packets flowing over the OpenFlow channels and possibly modifying them. These types of attacks have serious implications in IoT environments due to constraints on deploying traditional security mechanisms.

To detect and mitigate cyber threats against SDN networks, there are typically several different categories of detection and mitigation approaches. For example, traffic-based analysis monitors packet characteristics and flow patterns to detect anomalies indicative of malicious traffic. Similarly, threshold-based monitoring monitors the network's normal operation and will alert when abnormal traffic is detected. Finally, machine learning approaches including Long Short-Term Memory (LSTM) models

and Self-Organizing Maps (SOM) can adaptively classify the network traffic and distinguish between legitimate traffic and malicious traffic. Further still, hardware-based detection approaches monitor resource utilization (e.g., CPU and memory) to detect attacks that consume resources beyond normal operations.

However, integrating these detection and mitigation approaches into SDN controllers poses challenges, including increased computational overhead, particularly in resource-constrained IoT environments. Lightweight detection approaches, such as Bloom filters, provide efficient packet inspection and low memory requirements. Furthermore, distributed learning approaches, such as FL, enable collaborative threat detection across network entities while preserving user privacy and reducing the load on the central controller. These concepts form the foundation for understanding the current state of research contributions and to identify opportunities to improve the security of SDN-based IoT systems.

2.1 Related Work

The increasing adoption of SDN in IoT networks has prompted extensive research into detecting and mitigating cyber threats within these environments. In order to provide a systematic evaluation of prior contributions, Table 2.1 provides an overall summary of the literature organized by several key dimensions. Specifically, studies are categorized in terms of the type of attacks addressed by the studies (e.g., Denial of Service (DoS), DDoS, MitM, New Flow (NF)), while the specific network components examined in each study are noted via columns indicating the Testing Environment (TE), Connection Type (CT) and Network Elements (Host, Controller (Ctr), Switch (SW)). Each study is also classified with respect to its focus being either IoT specific scenarios or general SDN environments via the "IoT" column. Finally, the architectural methods used by each solution are indicated via columns for the three approaches to building a solution: Centralized (Cent.), Statistical-Based (Stat.) and Learning-Based (Learn.). Lastly, each study's identified limitations and mitigation techniques are captured in separate columns. These results from the structured comparisons demonstrate that there are multiple types of attacks being targeted, various mitigation techniques being developed and a number of limitations that exist across the existing literature.

The table highlights key threats including DDoS, DoS, MitM, and NF attacks, alongside corresponding mitigation approaches. Most studies focus on detecting attacks through network traffic analysis and specific algorithms, such as Bloom Filters in [14] and Self-Organizing Maps (SOM) in [24]. However, significant gaps remain in addressing hardware resource constraints, particularly in IoT environments where CPU and memory usage are critical factors.

DDoS attacks are among the most prevalent threats in SDN environments, capable of exhausting controller resources and causing service unavailability. The work in [1] and [6] examines the impacts of DDoS attacks, with [1] investigating controller failure risks due to SDN's centralized architecture, while [6] highlights system exhaustion as attackers flood the network with malicious requests. These attacks disrupt not only controller functionality but also degrade overall network performance, particularly problematic in bandwidth-constrained IoT networks. Flow table overflow poses

another frequent threat by flooding the data plane with excessive flow entries, reducing network throughput, and increasing packet loss. Y. Qian et al. [19] and H. Luo et al. [15] explore this vulnerability by simulating flow table overflow scenarios in OpenFlow-enabled switches, demonstrating how malicious actors exploit the limited capacity of flow tables to destabilize networks. Results from [19] illustrate how flow table overflow leads to packet loss and availability issues, emphasizing the need for effective flow management strategies.

MitM attacks, while less frequent, pose severe risks to data integrity and confidentiality in IoT systems. Researchers in [14] investigate the impact of MitM attacks on OpenFlow channels and propose the use of Bloom filters to detect packet modifications, showcasing the importance of lightweight detection techniques suitable for IoT's limited resources. Collectively, these threats underscore the necessity for enhanced security frameworks within SDN-IoT networks that can dynamically respond to various attack vectors.

Paper	AT				Network			HRA		Cent.	Stat	Learn	Mitigation	Limitation
	DDoS	DoS	MitM	NF	TE	CT	Ctlr/SW/Hst size	CU	MU					
IoT Focus														
[4]	✓	✗	✗	✗	🖵	🖧	-	✓	✓	✓	✓	✗	Counter-based DDoS Attack Detection (C-DAD) Framework	Threshold Challenges
[14]	✗	✗	✓	✗	🖵	🖧	-	✗	✗	✓	✓	✗	Bloom Filters	Extreme Cases
[23]	✗	✗	✗	✓	🖴	🖧	1/3/2	✗	✗	✓	✓	✗	Smart Security Mechanism (SSM)	Basline Calculations
[22]	✓	✗	✗	✗	🖵	🖧	1/30/60	✗	✗	✓	✓	✗	SDN sEcure COntrol and Data plane (SECOD) Algorithm	IP Address Spoofing Challenges
[2]	✓	✗	✗	✗	🖵	🖧	1/1/6	✗	✗	✓	✓	✗	Cosine Similarity based detection	Relied on simulated traffic
Non-IoT Focus														
[19]	✗	✓	✗	✗	🖵	🖧	1/1/20	✗	✗	✓	✓	✗	Eviction Algorithm	Increased Packet Loss
[15]	✗	✓	✗	✗	-	🖧	1/1/50	✗	✗	✓	✓	✗	Least Frequently Used (LFU) Algorithm	Packet Loss
[1]	✓	✗	✗	✗	🖵	🖧	-/4/30	✗	✗	✓	✓	✗	AMS Sketch Algorithm	-
[7]	✗	✓	✗	✗	🖴	🖧	-	✓	✓	✓	✓	✗	DosDefender	-
[6]	✓	✗	✗	✗	🖵	🖧	1/3/-	✗	✗	✓	✗	✓	SOM	-
[25]	✗	✓	✗	✗	🖵	🖧	1/1/20	✗	✗	✓	✓	✗	Routing Aggregation Algorithm	Architectural Changes
[24]	✓	✗	✗	✗	🖵	🖧	-	✗	✗	✓	✗	✓	SOM	False Positives
[17]	✓	✗	✗	✗	🖵	🖧	1/9/64	✗	✗	✓	✓	✗	Entopy Measuremnet on Destination IP	-
[11]	✗	✓	✗	✗	🖵	🖧	1/1/2	✗	✗	✓	✓	✗	Rate Limiting	-
[10]	✓	✗	✗	✗	🖵	🖧	1/2/8	✗	✗	✓	✓	✗	Statistical Defense Mechanism	-
[12]	✓	✗	✗	✗	🖵	🖧	2/10/20	✗	✗	✓	✓	✗	CDA Threshold-Based	-
[13]	✓	✗	✗	✗	🖵	🖧	2/10/20	✗	✗	✓	✗	✓	CDA Learn-Based LSTM	LSTM model relies on only two features
-	✓	✗	✗	✗	🖵	🖧	1/4/16	✓	✓	✗	✗	✓	CDA Threshold-Based / CDA Learn-Based LSTM	Limited to volumetric attacks, fixed thresholds

AT – Attack Types, **HRA** – Hardware Resource Analysis,
NF – New Flow, **TE** – Testing Environment (Emulator,Testbed),
CT – Connection Type (Ethernet),
CU – CPU Usage, **MU** – Memory Usage,
Cent. – Centralized, **Stat** – Statistical Based, **Learn** – Learning Based,
Ctlr – Controller, **SW** – Switch,
Hst – Host, 🖧 - Ethernet,
🖵 - Emulator, 🖴 - TestBed

Table 2.1 Literature Overview

Mitigation approaches proposed in existing literature focus on flow table management, traffic classification, and anomaly detection. Flow entry management strategies discussed in [15] and [25] address flow table overflow by dynamically adjusting flow entry timeouts and evicting less frequent entries. The work in [25] introduces a multilevel flow table architecture that combines Ternary Content Addressable Memory (TCAM) and Static Random Access Memory (SRAM) to expand flow table capacity without increasing power consumption. These techniques enhance controller performance during flow table overload, though their effectiveness is limited by IoT hardware constraints.

Traffic classification and anomaly detection techniques feature prominently across multiple studies. The approach in [6] uses SOM to classify network traffic, distinguishing between normal and malicious flows based on traffic characteristics, enabling administrators to receive alerts on potential threats for prompt intervention. Y. Xu et al. [24] and S. M. Mousavi et al. [17] employ entropy-based and statistical methods to monitor traffic anomalies. Specifically, [17] calculates the entropy of destination IP addresses to detect DDoS attacks early, achieving a 96% detection rate within the first 250 packets of attack traffic. However, these methods rely on significant computational resources and may struggle in resource-constrained IoT setups where frequent packet analysis can overwhelm system capacity.

Another promising approach, as demonstrated in [14], which uses Bloom filters to monitor packet integrity across flow paths, detecting MitM attacks with minimal overhead. This method leverages lightweight filters to identify discrepancies in packet forwarding, showing potential as a low-resource alternative to traditional methods. The authors in [10] propose SDNScore, a statistical defense mechanism against DDoS attacks that empowers SDN switches to autonomously classify and prioritize traffic flows based on predefined criteria. SDNScore outperforms conventional entropy models at distinguishing between malicious and legitimate traffic, though its effectiveness depends on sufficient computational power.

Building upon controller-based approaches, [12] [13] propose a CDA implemented directly on the SDN controller to enhance resilience against cyber threats in tactical networks. This CDA employs a dual-detection approach: threshold-based monitoring of flow entry counts and packet rates, combined with machine-learning-based detection using LSTM models to identify flow table flooding attacks. Upon detection, the CDA employs reactive and proactive response mechanisms, including clearing flow tables, blocking malicious ports, isolating compromised switches, and initiating failover to backup controllers to ensure continuous operation and maintain network integrity.

Despite these contributions, several limitations restrict the applicability of existing methods in SDN-enabled IoT networks. Most approaches rely heavily on traffic-based analysis, which can be computationally intensive. The authors in [4] and [2] highlight the need for efficient resource management in IoT networks, as excessive dependence on traffic monitoring can strain CPU and memory resources, especially in large-scale deployments. Moreover, many proposed methods are reactive, addressing threats only after they are detected. S. Wang et al. in [22] emphasize this drawback, noting that while SECOD is effective in reactive scenarios, it lacks proactive capabilities, limiting response speed in fast-evolving attack situations.

Adaptability to diverse IoT environments presents another challenge, as resource constraints and device heterogeneity can hinder threat detection. The work in [4] presents a C-DAD framework that, while effective in IoT contexts, is limited by its testing environment and protocol dependencies. This framework's accuracy in real-world IoT networks remains uncertain due to challenges with adapting predefined thresholds to dynamic traffic patterns. Similarly, [2] demonstrates that existing DDoS mitigation strategies often struggle to account for network fluctuations caused by resource limitations, highlighting the need for more adaptive solutions. A limitation identified in [12] [13] is that the flow table flooding LSTM model within their CDA currently relies on only two features, which can lead to misclassifications. Additionally, more comprehensive evaluation is needed to compare threshold-based and machine learning-based models, specifically by including cpu and memory usage metrics in test environments.

The table shows that while IoT-focused studies propose frameworks such as C-DAD [4], mitigation techniques often rely on thresholds that may not adapt well to dynamic IoT conditions. Non-IoT-focused studies explore flow management and statistical models but lack considerations for the constrained nature of IoT devices [12]. This comprehensive overview emphasizes the need for integrating hardware-level monitoring into existing threat detection frameworks to address the dual challenges of scalability and resource constraints in IoT networks. By combining CPU

and memory usage metrics with traffic data, enhanced detection capabilities become possible, as unusual hardware usage patterns could serve as early indicators of malicious activity. Such an approach would reduce the computational strain of current methods while providing more timely and accurate threat detection suitable for resource-constrained SDN-enabled IoT environments.

3

Design

Traditional methods for detecting intrusions using SDN and/or IoT environments cannot effectively detect all forms of attacks due to high traffic volumes on each device; a lack of prior experience with such a wide range of threats; and limited resources on each device. Also, traditional approaches to intrusion detection are mostly based on static models and centralized monitoring systems; therefore, they may not adapt quickly to new attack types. Moreover, collecting raw network data or system logs at a central location can lead to privacy issues, communication overhead, and single points of failure.

Therefore, the goal of this research is to propose a FCDA that leverages FL and unsupervised anomaly detection to support adaptive, privacy-preserving, and resource-aware intrusion detection in SDN-enabled IoT networks. The proposed system has been designed to run in two closely related but distinct stages: an offline stage and an online stage. The offline stage involves building a reliable initial detection model and preparing artifacts for deployment using historical data that was collected during both normal and attack scenarios.

In the offline stage, the focus is on preprocessing and normalizing the collected data, pretraining an autoencoder to build a detection model based on normal behavior, optimizing hyperparameters for training the detection model, and refining the global detection model through FL. As a result of the offline stage, there will be deployment ready artifacts, specifically the trained model weights, client specific training parameters, anomaly detection threshold values, and a normalized state that remains constant. These artifacts will serve as a baseine for deployment of the FCDA in the online phase on a variety of different IoT devices.

The online phase will represent the actual runtime execution of the FCDA in an operational SDN-IoT environment. During this phase, the client agents that have been installed on each of the IoT devices will continuously collect and monitor hardware and network level data, conduct real-time anomaly detection using reconstruction error, and optionally contribute to FL rounds. Each round of FL will be managed by the federated server that utilizes FedAvgM. The advantage of utilizing FedAvgM is

that it provides the ability to improve performance over time in a distributed manner and maintain performance stability despite the fact that the data being used for training is Non-Independent and Identically Distributed (non-IID).

An important aspect of the FCDA is that the continuous anomaly detection is independent of FL, thereby eliminating potential delay in intrusion detection caused by coordinating FL rounds. Figures 3.1 and 3.2 illustrate the entire life cycle of the proposed FCDA, from design-time model development to runtime execution and adaptation. Specifically, Figure 3.1 represents the pipeline for offline knowledge generation, depicting the major design decisions associated with data preparation, model training, optimization, and artifact creation. On the other hand, Figure 3.2 displays the workflow for the online detection, monitoring, and federated adaptation phases illustrating the relationships between the client agents, the federated server, and the decision-making processes for both anomaly detection and responses.

The separation of offline and online stages is a critical design choice. It enables the completion of computationally intensive operations (e.g., large-scale training, hyperparameter tuning) in the offline phase, whereas the online phase retains lightweight properties, making them feasible for constrained environments. At the same time, the inclusion of FL in the online phase provides the capability to update the model in response to changes in network conditions and attack behaviors without having to directly access raw data centrally.

This chapter describes the design of the proposed system. Section 3.1 provides additional information about the offline knowledge generation process, providing a detailed description of each component illustrated in the offline knowledge generation. Section 3.2 describes the online monitoring, detection, and FL components, focusing on the runtime behavior, decision logic, and overall scope of the system.

3.1 Offline Knowledge Generation

The Offline Knowledge Generation Stage generates an effective and deployable anomaly detection model before its deployment. It uses data previously collected in the target environment to learn the typical patterns of the system's behavior, tuning its model parameters and creating reliable anomaly threshold values. As such, by doing the computationally expensive work offline, the proposed system minimizes runtime overhead on resources constrained devices with consistent results across all deployed client devices. The artifacts produced during this offline pipeline serve as the basis for the online monitoring and the Federated Adaptation mechanisms shown in Figure 3.1.

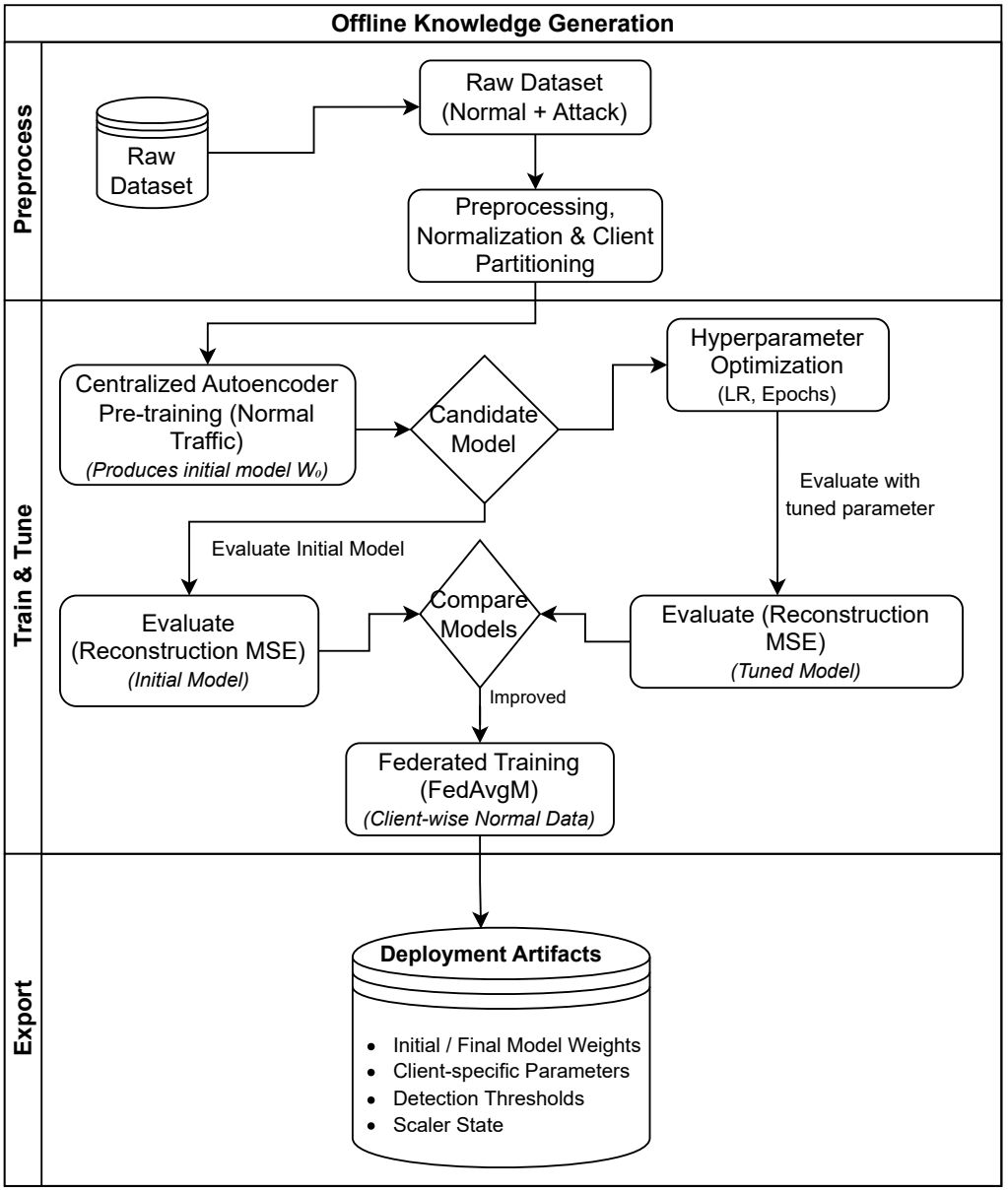

Figure 3.1 Offline Knowledge Generation.

3.1.1 Data Preparation and Preprocessing

Anomaly detection in resource constrained SDN-IoT systems is heavily reliant on the integrity and consistency of the input data. In order to achieve this, the proposed system employs a structured Data Preparation and Preprocessing pipeline to ensure that the features measured from various hosts are comparable, resilient to noise, and compatible with both Centralized and FL paradigms.

The raw data collected in the Offline Stage is comprised of system-level and network-level measurements taken from multiple IoT hosts under both normal and attack conditions. Each record represents a snapshot of host activity over a specified observation window and includes a combination of hardware utilization metrics (e.g., CPU usage, Memory usage, Process count, System load) and network traffic characteristics (e.g., Packet counts, Protocol distributions, Connection indicators). The hybrid design of these features allows the detection model to detect both computational stress and abnormal communication patterns, both of which are typical indicators of cyber attacks in SDN-enabled IoT networks.

Before the model training phase begins, the dataset is cleaned by removing incomplete, undefined or numerically unstable records. Specifically, records containing missing values, infinite values or undefined measurements are dropped to avoid bias and instability in the learning process. The selected feature set and essential identifiers are retained in the cleaning phase to ensure that the preprocessing pipeline remains lightweight and reproducible.

A global Min-Max Normalization Strategy is employed to ensure that all clients have consistent feature scaling. A single Min-Max Scaler is fitted to the combined feature space of the entire offline dataset and is then applied uniformly to all samples. This strategy guarantees that all features will be scaled to the same numerical range independent of the client from which the data was originated. Consistent scaling is particularly important in FL environments where heterogeneous scaling among clients can severely impair aggregation performance and slowing convergence.

Following normalization, the dataset is partitioned on a per-client basis to reflect the distributed nature of the target deployment environment. For each client, samples are divided into two subsets: one which contains only normal samples used for unsupervised model training and threshold estimation and a mixed subset containing both normal and attack samples reserved for later evaluation. The separation of samples assures that labels are not available during training and aligns with the intended unsupervised anomaly detection paradigm.

The output of this preprocessing stage is a collection of client-specific datasets having common feature representations and clearly defined roles in the subsequent training and evaluation phases. By performing normalization and partitioning offline, the proposed system ensures that all deployed clients have a common feature space and scaling policy, allowing for stable federated optimization and reliable anomaly detection during online operation.

3.1.2 Centralized Autoencoder Pre-training

A deep autoencoder is chosen as the core anomaly detection model because prior research [16] indicated it to be effective for intrusion detection within IoT environments, especially when little-to-no labeled attack data exists for a given environment. Moreover, the authors of [16] were able to demonstrate how, due to the consistent and predictable nature of IoT device traffic patterns, a deep autoencoder could effectively identify a large number of both known and unknown attacks based on the reconstruction error of the data fed into the autoencoder. Following this, [18] demonstrated that autoencoders can also naturally combine with FL, thereby providing a superior alternative compared to supervised models in heterogeneous IoT networks.

Autoencoders contain two primary functions, an encoder function $f\theta(\cdot)$, and a decoder function $g\phi(\cdot)$. An encoder function $f\theta(\cdot)$ is responsible for mapping a feature vector into a lower dimensional latent representation. In contrast, the decoder function $g\phi(\cdot)$ is responsible for reconstructing the original input feature vector from the latent space representation generated by the encoder function. If the input vector $x \in \mathbb{R}^d$ is passed through the encoder function $f\theta(\cdot)$, then the output will be a latent representation $z = f\theta(x)$ and the input vector can be reconstructed from the latent space representation z by passing z through the decoder function $g\phi(\cdot)$ to generate $\hat{x} = g\phi(z)$.

The parameters θ and ϕ for the encoder and decoder functions, respectively, are determined by minimizing the difference between the input feature vector and the reconstructed input feature vector, i.e., the reconstruction error, which is typically computed as the mean squared error (MSE) loss:

$$L(x, \hat{x}) = \sum_{i=1}^{d} (x_i - \hat{x}_i)^2$$

Prior to online operation, the autoencoder is trained only on normal data, thus enforcing the assumption that there is no labeled data for attacks during design time. This is beneficial because the autoencoder is able to learn a compact representation of normal behavior. As a result, any anomalies or malicious activity will cause a significant increase in the reconstruction error. This concept was demonstrated empirically in [16], where reconstruction-based detection was shown to achieve nearly perfect detection rates while simultaneously achieving a very low false positive rate for a variety of different IoT devices and attack types.

The proposed system utilizes a centralized pre-training strategy during the offline phase. Specifically, normal data collected from each client is combined to form a global model W_0 during the offline phase. There are two reasons why this design choice was made. Firstly, utilizing a centralized approach to pre-train the autoencoder allows the autoencoder to capture a larger distribution of normal behaviors from heterogeneous devices. Consequently, this will improve the ability of the autoencoder to generalize when the system is deployed. Secondly, utilizing a centralized pre-training approach provides a stable starting point for the FL process. As such, the convergence time of the FL process is improved, and the instability that is common during the first few rounds of FL is reduced. While [18] utilized a completely decentralized training approach for the autoencoder, a common approach to improve robustness in federated systems that utilize non-IID data distributions is to use a centralized warm-start pre-training strategy.

The autoencoder employed in this study uses a symmetric encoder-decoder structure with progressive decreases in the dimensionality toward a single bottleneck layer. To enable modeling of complex relationships between system and network features while also being computationally efficient enough to operate in resource constrained environments, nonlinear activation functions are employed. Due to the information compression effect of the bottleneck layer, the autoencoder retains only the most critical characteristics of normal behavior.

Once the autoencoder has completed centralized pre-training, the model parameters W_0 represent a baseline model that is capable of detecting anomalies through

the calculation of reconstruction error. The parameters W_0 are not used to make predictions or for detection, but rather they serve as a basis for hyperparameter optimization and federated training in subsequent phases of the offline pipeline. This staged approach is based upon the results presented in [18], which demonstrated that autoencoders can adaptively be trained under FL conditions while maintaining their unsupervised detection abilities.

The centralized autoencoder pre-training methodology provides a scientifically justified and empirically supported foundation for the proposed FCDA. Furthermore, by employing reconstruction-based anomaly detection, the proposed system benefits from the properties demonstrated by previous IoT Intrusion Detection System (IDS) research, specifically independence from labeled attack data, the resilience to previously unseen threats, and the potential to utilize FL while expanding these capabilities to include the deployment in a SDN-IoT enabled network.

3.1.3 Hyperparameter Optimization

Centralized pretraining provides a good starting point for an autoencoder's initial model, but the performance and reliability of an autoencoder particularly in a FL environment depend heavily on its training parameters. Training parameters including the learning rate and the number of training epochs can have a direct effect on how quickly the model will converge, how accurately it will reconstruct data, and the potential for the model to fit too closely to the normal behavior of local data. Because data distribution and available resources vary across IoT devices, fixed hyperparameter values may produce suboptimal or unreliable performance once the model is deployed in a heterogeneous IoT environment.

To address this issue, the proposed system includes a dedicated hyperparameter optimization stage within the offline knowledge generation to optimize the training parameters of the autoencoder prior to deployment in a FL setting. The hyperparameter optimization stage uses a systematic exploration of the candidate hyperparameter values to find those values that result in minimum reconstruction error while also providing adequate representation of all clients normal data.

The optimization workflow begins with a centrally pre-trained model W_0. Multiple training trials of the autoencoder are performed using different hyperparameter configurations. For each configuration, the autoencoder is trained on normal data, and then evaluated based on the reconstruction error metric as defined in Section 3.1.2. The goal of the optimization process is to identify the hyperparameter configuration that results in the lowest average reconstruction error thereby enabling the model to best capture the behavior of normal conditions.

The hyperparameter optimization problem can formally be stated as:

$$\lambda^* = \arg\min_{\lambda \in \Lambda} \mathbb{E}\big[L(y, f(x; \lambda))\big]$$

where $L(\cdot)$ is the reconstruction loss and the expectation is taken over normal samples. This formulation aligns with the unsupervised anomaly detection objective and avoids reliance on labeled attack data.

The decision to perform hyperparameter optimization offline was made deliberately. Hyperparameter tuning is computationally intensive, and therefore cannot be done on resource constrained IoT devices during runtime. However, by performing the hyperparameter optimization offline, the proposed system ensures that the online component of the system remains lightweight, while still utilizing training parameters that were determined through careful selection. Decoupling the hyperparameter optimization from the deployment process also allows for better reproducibility and facilitates controlled experimentation during the design phase.

Another aspect of the design involves the relationship between hyperparameters and FL dynamics. Previous research [18] indicates that autoencoders trained under FL settings are sensitive to overly aggressive learning rates and excessive local training, especially when client data is non-IID. Therefore, the hyperparameter optimization stage focuses on conservative parameter selections that support stable, smooth convergence across clients when aggregated. The selected hyperparameters will serve as a baseline for subsequent federated training and enable further adaptation during online operation.

As a result of the hyperparameter optimization stage, a set of valid training configurations are produced that demonstrate improved reconstruction performance. The valid training configurations are then passed to the model comparison stage where they are compared to the original pre-trained model to evaluate if there was a meaningful improvement in reconstruction quality. Only the training configurations that provide significant improvements in reconstruction quality will be retained for federated training and deployment.

To sum up, hyperparameter optimization is crucial in bridging centralized pretraining and FL. Through systematic refinement of training parameters offline, the proposed design increases the reliability of the model, decreases the likelihood of unstable updates during federated aggregation, and ensures that the deployed clients are operating with optimized parameters for performing unsupervised anomaly detection in heterogeneous SDN–IoT environments.

3.1.4 Federated Training with FedAvgM

After the initial centralized pre-training and hyperparameter selection phases, the proposed system proceeds to a federated training phase to fine tune the intrusion detection model, while keeping the data local to each client and protecting client privacy. This second stage was motivated by the fact that, due to their distributed and heterogeneous nature, IoT environments present different traffic patterns and resource conditions to each individual device. Unlike traditional approaches which collect the raw data at a central site, FL allows multiple models to be improved collaboratively by exchanging model updates, thus minimizing the amount of communication required between clients and servers, and consequently the risks to client privacy.

In this study, FedAvgM has been chosen as the aggregation method. The motivation behind choosing this specific aggregation method comes from the findings of several other studies that have shown that standard Federated Averaging (FedAvg) performs poorly under non-IID client data distributions, a situation common

in IoT applications. For example, Olanrewaju-George et al.[18] uses FedAvgM for federated intrusion detection, and it finds that using FedAvgM leads to better convergence stability and less false positives when training unsupervised autoencoders on heterogeneous IoT devices. Additionally, Hsu et al.[9] systematically analyzes how non-IID client data affects FL, and it empirically shows that including server-side momentum in FedAvg helps reduce oscillations and slow convergence that are often associated with FedAvg.

For clarity, let W_t represent the global model parameters at federated round t, and let $W_{t+1}^{(k)}$ represent the locally updated model parameters produced by client k after local training. In standard FedAvg, the global model update is computed as a weighted average of all client updates:

$$W_{t+1} = W_t + \sum_{k \in \mathcal{K}_t} \frac{n_k}{\sum_j n_j} \left(W_{t+1}^{(k)} - W_t \right)$$

where $\mathcal{K}_t$ represents the set of clients involved in federated round t, and n_k represents the number of local samples that client k uses to train its local model.

FedAvgM modifies this basic formula by adding a server side momentum term that accumulates the update direction over time. First, an aggregated update ΔW_t is computed as follows:

$$\Delta W_t = \sum_{k \in \mathcal{K}_t} \frac{n_k}{\sum_j n_j} \left(W_{t+1}^{(k)} - W_t \right)$$

Then, a momentum vector M_t is updated as follows:

$$M_t = \beta M_{t-1} + \Delta W_t$$

Where $\beta \in [0, 1)$ is the momentum coefficient that controls the impact of past updates. Finally, the global model is updated based on:

$$W_{t+1} = W_t + M_t$$

The accumulation of historical update information in FedAvgM helps smooth out sudden changes in model parameters that occur due to conflicting client updates or large variations in the local gradients among clients. Given the narrow and possibly biased views of normal behavior that each device may experience in an IoT environment, this smoothing property is especially useful for intrusion detection in such environments, as demonstrated in [18] by the use of FedAvgM for federated autoencoder training resulting in more stable reconstruction error distributions and lower false positive rates than those achieved with FedAvg. Furthermore, [9] supports this result by showing that FedAvgM achieves consistent convergence even in cases with extreme non-IID data partitions and partial client participation.

Offline federated training is performed in the proposed architecture to generate a refined global model that captures the heterogeneity among clients and is stable. Hyperparameters optimized in the previous stage are applied during federated training to ensure that the updates made by each client are localized and do not diverge.

It should be noted that this offline federated training does not eliminate the need for online adaptation. Instead, it creates a robust global model that can be safely deployed and further adapted during run-time.

Therefore, the output of this federated training stage is a globally optimized model that includes both the centralized knowledge and the federated refinement under non-IID data conditions. Together with the optimized hyperparameters and the normalization state of the model, this model is the foundation upon which the deployment in the online monitoring stage will take place. By explicitly addressing the issues created by non-IID data through the use of FedAvgM, the proposed system aligns itself with current state-of-the-art FL methods and is suitable for anomaly detection in SDN enabled IoT systems.

3.1.5 Deployment Artifact Generation

The final step of the offline knowledge generation pipeline involves creating a collection of "deployment artifacts" which allow for an identical and very light-weight initialization of the proposed system when it is operational online. The goal of this approach is to export the least amount of information necessary to accomplish inference, detect anomalies and perform controlled online adaptations rather than export entire training pipelines or model specifications as was done with prior approaches.

The first piece of information exported by the offline process is the initial model parameters that were generated after both central pre-training and federated refinement. These parameters represent the trained weights of the autoencoder and will be used as a starting point for runtime inference and any subsequent federated updates. By only exporting the learned weights, the proposed system avoids the unnecessary overhead while maintaining full compatibility with the autoencoder architecture defined on each client.

The second piece of information exported by the offline process is the normalization state created through the offline preprocessing. The normalization state includes the parameters necessary to apply Min–Max normalization in the same manner across all clients to ensure that the feature vectors observed during the online monitoring period are mapped to the same numeric space as those used during the offline training period. It is important to maintain a consistent normalization strategy because inconsistent feature scaling strategies may result in less reliable anomaly score values.

The third piece of information exported by the offline process is a set of client-specific training and detection parameters, such as the learning rate, the number of local training iterations and the anomaly detection threshold value for each client. The values of these parameters are determined during the offline optimization period to take into consideration the client-side heterogeneity in behavior and resources. By determining parameters on a per-client basis, the proposed system provides controlled and stable online adaptation while preventing a global parameterization.

In total, the artifacts provided by the offline process create a deterministic and reproducible initialization state for the online monitoring process. Each client agent loads the initial model weights, applies the previously defined normalization state, and operates under the previously validated client-specific parameters from the start.

This design allows for immediate initiation of anomaly detection upon deployment of the system without having to conduct local retraining or recalculate parameters, and provides a well-defined and stable baseline for the subsequent FL.

By defining offline knowledge in terms of lightweight deployment artifacts, the proposed system creates a distinct separation offline optimization and online execution. This separation results in reduced online computational overhead, improved robustness in resource constrained environments, and a reliable foundation for the processes related to online monitoring, detection and federated adaptation discussed in the next section.

3.2 Online Monitoring, Detection, and Response

The online stage of the proposed system represents the run-time operation of the FCDA within a live SDN-IoT environment. As such, it differs from the offline stage in that the focus is placed on continuous monitoring and real-time anomaly detection as opposed to model construction and optimization. Additionally, due to resource constraints, the online stage places an emphasis on adaptive learning. Designed for autonomous operation, the online stage utilizes deployment artifacts generated during the offline stage to provide immediate functionality upon startup.

Upon startup, lightweight client agents deployed on IoT Hosts continuously collect system level and network level metrics, normalize those metrics through a fixed normalization policy. Then apply those metrics to the autoencoder model deployed at each host site. As a result, the anomaly detection mechanism operates in real-time utilizing the reconstruction error metric thereby enabling the system to detect deviations from learned normal behavior independent of labeled attack data. Moreover, the anomaly detection mechanism operates independently of FL rounds, thereby providing continuous security monitoring regardless of training coordination.

In addition to online federated adaptation, the system also allows for collaborative refinement of the detection model among the client agents. Selective and conditional local training is allowed, provided there is availability of high-confidence normal samples and server side client selection. Updates to the detection model are aggregated by the central federated server through the use of FedAvgM and subsequently distributed back to the clients, thus allowing for gradual adaptation to evolving workloads and network conditions while maintaining data locality.

Figure 3.2 represents the online monitoring and FL workflow, illustrating the interaction between the client agents, the federated server and the decision logic utilized to make anomaly detection decisions. The remainder of this section will detail the design of the online stage of the system, including but not limited to system deployment assumptions, continuous monitoring and feature extraction, anomaly detection mechanisms, FL coordination and response scope.

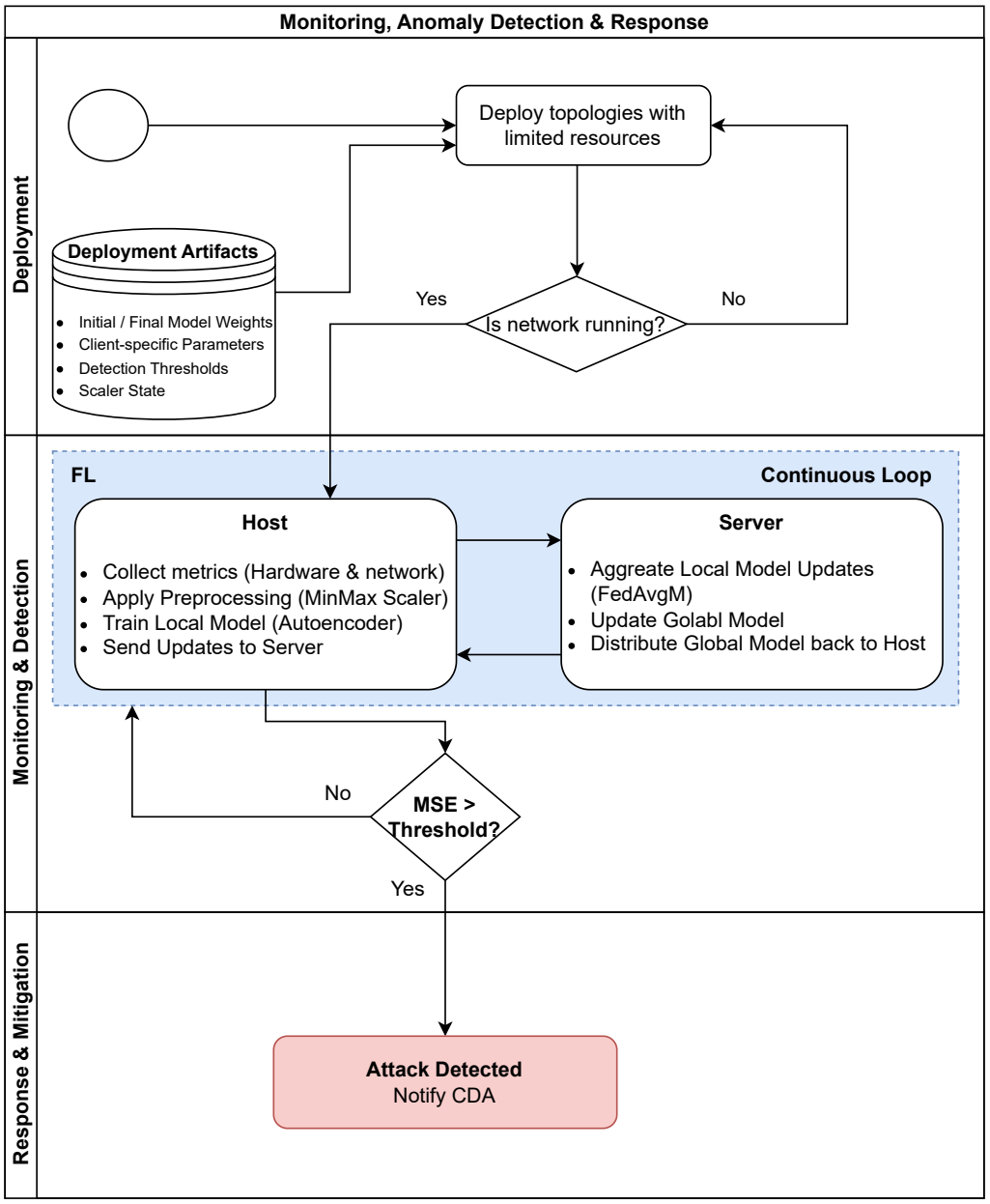

Figure 3.2 Online Training.

3.2.1 System Deployment in MininetFed

The online stage of this proposed system will be executed on an emulated environment using a MininetFed [21] based simulation environment, to provide a reliable

testing of FL in SDN and resource constrained environments. MininetFed adds to Mininet the support for FL and containerized hosts with configurable resource limits, to model the deployment of SDN enabled IoT systems.

In the proposed implementation, every IoT device is modeled by a lightweight containerized host, and a client agent runs on each of them. Each client agent has three main functions, namely monitoring, anomaly detection and training of the local model. Every client is run inside a container with very limited computing resources like CPU shares and memory, in order to represent the operational limitation of real world IoT devices. Since there are no resource limitations enforced at deploy time in the proposed design, the operational resources of each component in the online part of the system can be evaluated in the most realistic way possible.

Next to the client hosts, a federated server is deployed to manage the FL rounds. The server is responsible for the client registration, for selecting the participants for a round of FL, for aggregating the local model updates from the participants, for distributing the updated global model parameters to the participants. However, the server never accesses the raw monitoring data gathered by the clients. All the interactions between the clients and the server are limited to exchanging model updates and additional training metadata. This design choice preserve the data locality and thus complies with the objective of preserving privacy of FL.

To enable the communication between the different parts of the system, a message oriented publish-subscribe protocol is used. This communication layer allows the clients to work asynchronously and independent of each other, and allows the clients to continue their monitoring tasks and participate in federated training rounds whenever they want, and whenever they have the possibility to do so. Therefore, the monitoring of anomalies continues to run even if the federated training is delayed or temporarily unavailable.

Additionally, to the components described above, the proposed system includes a logically separated SDN control plane. Although the proposed system does not enforce directly any mitigation action at the controller level, the proposed system is compatible to integrate such a functionality. Thus, the proposed system separates clearly the responsibilities of the control plane of the network from the responsibilities of the detection and response plane, to allow the integration of such a functionality in the future to automate the enforcement of mitigation actions without modifying the core detection and learning functionalities.

When the client agents are initialized, they load the deployment artifacts produced during the offline phase, i.e., the initial model weights, the normalization parameters and the client specific training and detection configurations. This initialization will enable the client agents to initiate the detection of anomalies immediately upon the activation of the network, without requiring an additional centralized training phase. As subsequent data becomes available, the models will then be progressively refined via FL.

Therefore, the deployment of the proposed system in a MininetFed based environment, offers a controlled but realistic scenario for the execution of the proposed online design. The combination of the use of containerized IoT hosts, a federated coordinator server and a communication layer allows the system to model the op-

erational constraints and the distributed nature of the SDN-IoT networks, while supporting continuous monitoring and adaptive learning.

3.2.2 Continuous Monitoring and Feature Extraction

Client agents are always monitoring both the network level and the host level of their system's behavior in real-time to enable timely and context aware anomaly detection. Since typical IoT devices are subject to limited CPU and memory, excessive or intrusive monitoring can cause a degradation of normal operation, exhaust resources that affect the monitoring process, or even cause an additional resource consumption. For this reason, the client agent monitoring process was developed to be lightweight and non intrusive, while at the same time obtaining enough information to adequately characterize both normal and abnormal system activity.

The host level of monitoring is focused on metrics which describe the computational state of the device. Metrics such as CPU utilization, memory usage, process activity, and short term system load, will help identify whether a system is experiencing resource starvation, an abnormal execution pattern, or a sudden increase in workload. Collecting the above mentioned metrics directly from the client agent's operating environment allows the system to detect local effects that would be difficult to detect via network observation.

Simultaneously, network level monitoring captures characteristics of the traffic of the system. Instead of examining packets in detail or keeping track of flows, the proposed design will monitor the system's traffic passively, aggregating the collected statistics in fixed duration time windows. Statistics collected include packet count, protocol distribution, diversity of sources and destinations, and connections. Since the agent does passive monitoring there is no additional traffic introduced by the monitoring system and the normal operation of the network should remain unchanged.

Feature extraction occurs over short, fixed duration time windows. Each window produces a small feature vector that contains a combination of the host and network measurement features. Each feature vector represents one snapshot of the system's behavior during a given time window and is the input to the anomaly detection model. The selected feature set has balance of expressiveness and efficiency to allow the anomaly detection model to detect correlated anomalies in both the computation and communication dimension without adding too much run time cost.

Each feature vector extracted is normalized by the fixed normalization state created during the offline phase before it is sent to the anomaly detection model for evaluation. Normalization is used to ensure that the training and inference distributions of the anomaly detection model are similar and prevent large scale differences from affecting the results of the reconstruction error. Normalization is applied locally and deterministically so that the system behaves uniformly across all clients regardless of their hardware differences.

Monitoring and feature extraction are decoupled from the FL process, ensuring that anomaly detection remains operational regardless of a client's participation in training rounds. This decoupling of monitoring from learning is a key design decision and provides the system with the flexibility to perform security monitoring while allowing for the concurrent development of the models.

In summary, continuous monitoring and feature extraction are the primary inputs to both the anomaly detection model and the FL model. The proposed design provides real-time situational awareness and is capable of functioning within the constraints of an SDN-IoT environment due to the use of low-weight host level metrics, passive network observations, and consistent normalization.

3.2.3 Anomaly Detection Mechanism

The proposed system utilizes an autoencoder based anomaly detection method, which utilizes an unsupervised approach to learn the normal behavior of the system. During operation, the model reconstructs input feature vectors and uses a reconstruction-based anomaly score to quantify deviations from learned normal patterns. Detection occurs locally at each client and is independent of the FL process, ensuring quick anomaly identification without relying on training schedules or communication delays.

Each monitoring cycle, a client agent will create a normalized feature vector $x \in \mathbb{R}^d$ (where d represents the number of monitored features). The normalized feature vector will be processed by the deployed autoencoder model to produce a reconstructed version of the normalized feature vector $\hat{x}$. The anomaly score is defined as the mean squared reconstruction error:

$$A(x) = \frac{1}{d} \sum_{i=1}^{d} (x_i - \hat{x}_i)^2$$

This score measures the degree to which the actual behavior of the system corresponds to the normal operating conditions that were learned during the offline phase of the development of the autoencoder. Since the autoencoder was developed only with normal data during the offline phase, normal observations should have low reconstruction errors, while any malfunctions or attacks could produce larger errors.

Anomalies can be detected based on a threshold-based decision rule. Let τ represent the anomaly detection threshold established during the offline phase of the development of the system. Then the decision regarding whether a particular observation x represents normal or anomalous behavior is as follows:

$$\text{Decision}(x) = \begin{cases} \text{Anomalous}, & \text{if } A(x) > \tau, \\ \text{Normal}, & \text{otherwise.} \end{cases}$$

This decision rule allows the system to immediately raise an alarm when it detects any suspicious behavior, without any need to consider past history or to process the observations in batches. Additionally, since the threshold value is established initially before deploying the system, the system will exhibit deterministic and replicable detection behavior for all clients at start up.

To provide stable adaptive behavior during online use of the detection mechanism, it also differentiates between normal behaviors with a very small reconstruction error, and those that fall in an intermediate region (borderline cases). If the reconstruction

error associated with a particular observation is significantly less than the threshold value, the observation is assumed to be a reliable example of normal behavior, and the observation may be saved to support local training during FL, as outlined in Algorithm 1. This design helps to prevent the inclusion of corrupted or uncertain samples into the training process, thus maintaining the integrity of the model.

Require: Normalized feature vector x, autoencoder model $f(\cdot)$, threshold τ
Ensure: Detection decision and optional training sample update

Receive normalized feature vector x
Compute reconstruction $\hat{x} \leftarrow f(x)$
Compute anomaly score $A(x) \leftarrow \text{MSE}(x, \hat{x})$
if $A(x) > \tau$ **then**
    Raise anomaly alert
    Mark sample as anomalous
else
    Mark sample as normal
    if $A(x) \leq \alpha \cdot \tau$ **then**
        Add x to local training buffer
    end if
end if

Algorithm 1 Anomaly detection at client agent

The confidence factor $\alpha \in (0, 1)$ controls how strictly normal samples are selected for local training.

Since the anomaly detection mechanism is lightweight and executed locally at each client, it requires only a single forward pass through the autoencoder per observation. This results in low computational overhead and enables real-time anomaly detection even on resource-constrained devices, making the approach suitable for continuous deployment in SDN-IoT environments.

In summary, reconstruction-based anomaly detection provides a principled and efficient basis for detecting anomalous behavior in SDN-IoT systems. The combination of fixed threshold values, confidence-aware sample selection, and continuous operation ensures the reliability of the detection mechanism while allowing for safe online adaptation through FL.

3.2.4 Online Federated Adaptation

To allow the continual update of the detection model as the network changes, the proposed system leverages online federated adaptation as a background process that runs alongside the real-time anomaly detection. In contrast to the communication round process seen in traditional FL workflows, the proposed design opts for conditional and opportunistic training that takes advantage of safe, informative learning opportunities that avoid wasting resources where possible.

3.2.4.1 Conditional Local Training

Local model training on each client is performed under conditional statements around observables and confidence in behavior. Rather than training on every observation the system observes, we selectively accumulate feature vectors classified as high confidence normal by the anomaly detection process in Section 3.2.3. Training on false or suspiciously ambiguous observations would pollute local training.

Rather than train on every observation however, we condition local training on accumulation of a sufficient number of observations such that training to statistical meaningfulness is done, and only where previous high-confidence confirmation can be relatively assured not to be a spurious event. Doing so reduces risk of chaotic updates when various observations are sparse or extreme, a process particularly risky in intrusion detection. Inclusion of any attack traffic risks degrading model accuracy severely. Local training is therefore performed at intermittent intervals and its retention in the buffer cleared after every training round.

3.2.4.2 Client Selection

Not all clients participate in all rounds of FL. On the server's side, there is a mechanism for client selection which specifies which update-contributing clients are allowed in every FL round. This technique makes FL accommodating to heterogeneous availability and resource conditions, and asynchronous client operation.

By separating monitoring from active training participation, our design insures that the clients are protected even when they are not selected for FL. Client selection reduces the communication required and avoids establishing an uncertain aggregation of poor-quality updates.

3.2.4.3 Server Aggregation Loop

The federated server orchestrates each round of training by harvesting local model updates and then redistributing the updated global model. This is another continuous server loop, again running without regard to any anomaly detection and monitoring activities running on the clients.

Each round, the server waits for updates from clients participating in the round. It aggregates only the updates tied to "valid" local training. Clients that did not meet training conditions or contributed only zero effective samples are not aggregated, in order to keep their "damaging" updates from disturbing the global update process. After aggregation, it broadcasts the updated global parameters out to all the clients.

This asynchronous and serially iterative aggregation loop operation allows the system to adapt incrementally and without strict synchronization across clients.

3.2.4.4 Connection to FedAvgM

Aggregation is performed using federated averaging with server momentum, as described in the offline design stage 3.1. In online adaptation, FedAvgM is used to

stabilise the global updates from the clients, mitigating issues that arise from heterogeneous, possibly non-IID client data. By incorporating information from historical updates at the server, FedAvgM reduces the impact of noise and abrupt parameter changes caused by diverging local updates.

Importantly, the online adaptation phase utilizes the same aggregation principles that were proven to be valid in the offline phase, thereby providing conceptual consistency between design time training and runtime learning. This continuity enables the system to take advantage of FL while exhibiting predictable and controlled behavior.

3.2.4.5 Stability Safeguards

To further enhance robustness during online adaptation, the proposed design incorporates several conceptual stability safeguards. First, local updates are scaled conservatively to prevent excessive deviation from the current global model. Second, aggregation explicitly excludes updates from clients that lack sufficient training data, reducing the influence of unreliable contributions. Third, the system maintains a fallback mechanism that allows clients to revert to a previously stable model state if abnormal behavior is detected following an update.

These safeguards collectively ensure that online federated adaptation improves the detection model gradually and safely, without compromising the reliability of anomaly detection. By prioritizing stability over aggressive optimization, the proposed design aligns with the operational requirements of security-critical SDN–IoT environments.

3.2.5 Response Strategy and System Scope

The response strategy of the proposed system is designed to support reliable anomaly detection and adaptive learning while maintaining a clearly defined scope of responsibility. Once the FCDA identifies anomalous activity at a client, the system generates an alert indicating the presence of a potential attack or abnormal behavior. The alert represents the end product of the detection process and also serves as the connection between the proposed FCDA and the external response mechanism.

Instead of attempting to enforce the mitigation of the identified threat, the FCDA will forward the alerts from the detection component to an existing CDA. The CDA as described in prior research, is a policy aware and contextually driven mitigation component capable of performing actions such as changing flow rules, switch isolation, controlling traffic rates, or reassigning controllers. By forwarding the alerts to the CDA instead of attempting to perform the actual mitigation, the proposed design eliminates duplicated response logic and ensures that the identified threats are acted upon in a timely and controlled manner.

As stated previously, the separation of the detection from the response mechanisms is purposeful. In order to effectively respond to anomalies in SDN-IoT environments, response decisions are made based on multiple variables including; network policy, operational environment, and the level of risk associated with a particular action. Therefore, making automatic mitigation decisions based solely on the outputs of

the detection process may result in risks including the interruption of normal traffic flows and potentially negative consequences of false positives or transient anomalies. By isolating the detection and learning processes of the FCDA and delegating the decision-making authority for response actions to the CDA, the proposed design minimizes the occurrence of premature and/or potentially harmful responses to anomalies.

In terms of architectural considerations, the proposed design allows for the seamless interaction between the FCDA and the CDA. The structured input provided by the FCDA to the CDA as a result of the detection of anomalies, enables the development, refinement, and deployment of response strategies independent of the detection and learning mechanisms of the FCDA. As a result of this modular architecture, the proposed design supports a high degree of adaptability across various deployments and enables response policies to evolve without the need for changes to the underlying detection framework.

In summary, the establishment of the system boundaries, as defined in the proposed design, enhance the reliability and usability of the proposed system in security critical environments. The FCDA retains the responsibilities of continuous monitoring, anomaly detection, and model refinement using FL. In contrast, the CDA retains the responsibilities of network control and enforcement. The clear definition of responsibilities provides scientific rigor and adheres to recognized best practices for designing secure SDN architectures.

4

Evaluation

4.1 Motivational Analysis: Impact of Network Attacks on Host Hardware Resources

Before designing and implementing the FCDA, we conducted an initial investigation of the relationship between network based attack and hardware resource utilization at the host level. The preliminary investigation was based upon the understanding that most IoT and Edge devices are constrained in terms of their available hardware resources, and that if an excess amount of these hardware resources are being utilized; it could be considered a type of service degradation or denial, even though the network may not be completely saturated.

In order to examine this relationship empirically, controlled volumetric attacks were run in an emulated environment and hardware resource metrics were collected from the individual hosts. The metrics that were collected were the CPU usage and memory usage of each host during both normal operation, and while executing SYN flood, ICMP flood, and UDP flood attacks. These attacks were chosen as they represents common forms of volumetric DoS attacks.

Figure 4.1 demonstrates the CPU and memory usage of hosts over time during a SYN flood attack. During the attack, CPU usage increased rapidly and remained at a high level for the duration of the attack window, returning to baseline levels shortly after the attack concluded. However, memory usage did not follow the same pattern and showed a gradual rise in usage that remained elevated for a much longer duration and continued to remain elevated long after the attack traffic had stopped.

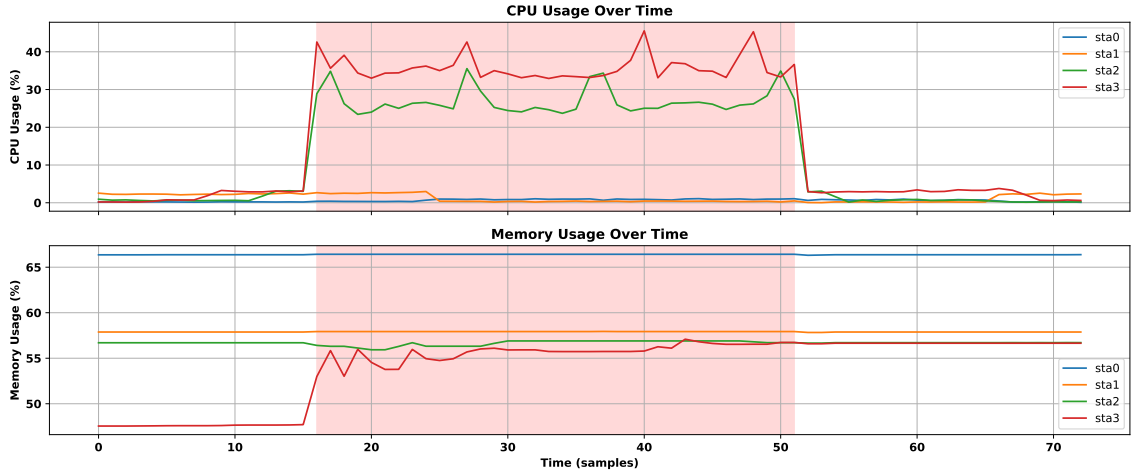

Figure 4.1 SYN Flood

In comparison, ICMP flood attacks have a greater negative effect on the host as shown in Figure 4.2. In the ICMP flood attack, CPU usage climbed very rapidly to nearly maximum capacity on affected hosts for extended durations. This behavior is consistent with the large amount of processing required to handle the high rate of ICMP traffic being sent to the host, and indicates a high likelihood of disrupting services to the host under sustained attack conditions. Memory usage also continued to climb steadily throughout the attack window and remained elevated, which demonstrates that ICMP floods cause continuous computational and memory stress to host devices.

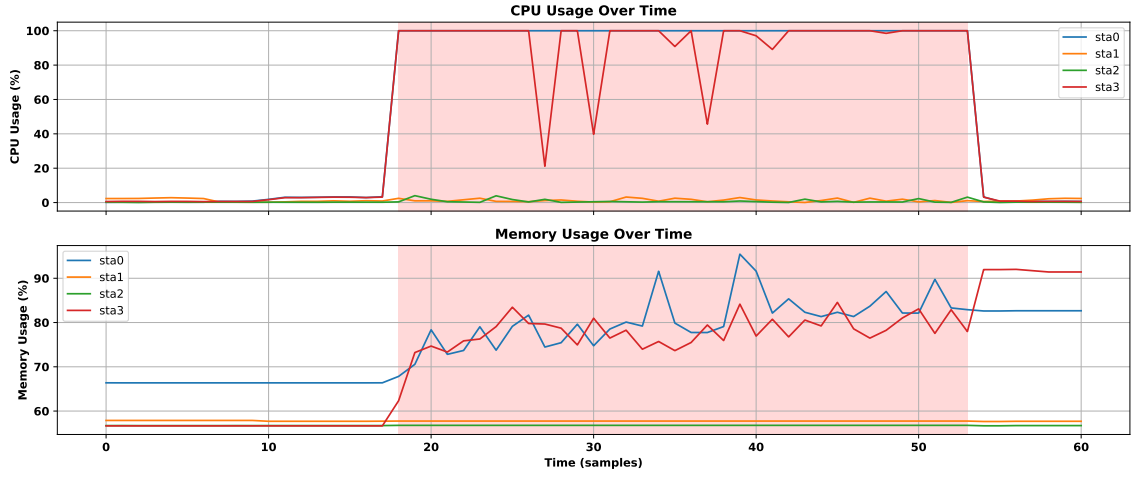

Figure 4.2 ICMP Flood

As indicated in Figure 4.3, CPU usage during the execution of a UDP Flood Attack did not reach the same extreme levels seen in the ICMP Flood Attack, but it was still significantly higher than the baseline for the duration of the attack. Memory usage again demonstrated a consistent upward trend in usage throughout the attack window, which reinforced the conclusion that volumetric network attacks consistently create non-trivial memory requirements for host devices, irrespective of the protocol used to conduct the attack.

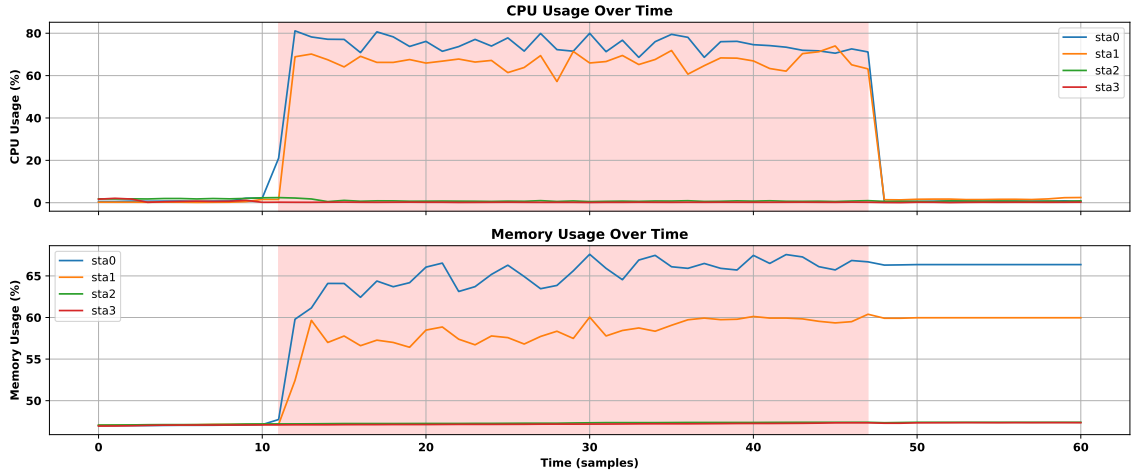

Figure 4.3 UDP Flood

Collectively, the Figures clearly show a strong association between volumetric network attacks and host-level hardware stress. CPU usage appears to respond very rapidly to attack traffic and then returns to its baseline state. On the other hand, memory usage continues to grow throughout the attack and may remain elevated for a much longer period of time than CPU usage. Furthermore, the variability in resource impacts among hosts even in identical attack conditions, which further emphasizes the heterogeneity present in distributed environments.

These findings have provided important insights regarding the shortcomings of network-only intrusion detection techniques. Network traffic patterns can reveal anomalies in packet behavior, but they cannot identify the stress that individual devices experience as a result of attacks. Therefore, as a result of the lack of direct access to per-host CPU and memory usage by the controller in SDN-IoT environments, centralized detection is insufficient for detecting resource degradation due to attack.

Figures 4.1-4.3 provided the empirical basis for the proposed FCDA framework. By allowing each client to collect both network and hardware metrics locally, and by having clients share learned representations instead of raw data through FL, the proposed solution addresses both the visibility and scalability issues. This analysis provides empirical justification for including host-level hardware resource awareness into distributed intrusion detection systems, especially those deployed in resource constrained IoT environments.

4.2 Experimental Environment and Setup

In this section, we describe the experimental setup and environment of the proposed FCDA, which will be evaluated based on experiments carried out with a controlled emulation platform to guarantee the reproducibility of results and accurately represent the operational constraints present in SDN-based IoT systems.

4.2.1 MininetFed Testbed Configuration

The experimental evaluation was conducted using MininetFed [21] version 1.0.2, executed on a local machine. MininetFed provides a testing environment to emulate SDN-based IoT systems that include both SDN, containerized hosts, and FL workflows; thus, allowing us to perform evaluations of distributed learning-based security mechanisms under constrained environments.

Ubuntu 20.04.6 LTS was installed as the host operating system, and containerized components were created using Docker version 28.1.1. Each of the emulated IoT hosts and the federated server were instantiated as Docker containers to enable detailed control over computational resources, including CPU shares and memory limits.

Asynchronous, decoupled communication between distributed components was achieved through an MQTT-based publish–subscribe mechanism. Mosquitto, the MQTT broker, was also included with the MininetFed framework. When the experiment began, the broker was run in a separate Docker container. No manual configuration or instantiation of the broker was required; rather, it was automatically managed by MininetFed during the experimental workflow. This allowed continuous, uninterrupted communication between client agents and the federated server during each experiment.

Component	Description
Emulator	MininetFed v1.0.2
Execution Mode	Local machine
Operating System	Ubuntu 20.04.6 LTS
Container Engine	Docker 28.1.1
Communication	MQTT (Mosquitto, via MininetFed)

Table 4.1 Testbed Configuration

4.2.2 Network Topology and Resource Constraints

All experiments used a single-switch SDN topology consisting of one OpenFlow switch and four host nodes, each representing an IoT client. This allowed us to test how resource restrictions affect both the learning dynamics and resource limitations while still providing centralized SDN control via a single forwarding element.

A remote SDN controller based on Ryu was used to manage the network. The Ryu controller was run outside of the MininetFed environment and communicated with the emulated switch through the control channel. This setup is similar to real-world SDN deployments where the controller runs separately from the data plane and client hosts.

To evaluate the behavior of our proposed system under different resource levels, we ran numerous experiments with varying CPU and memory constraints applied to the client containers. We used Docker CPU shares to limit CPU utilization, and we placed memory limits on the containers to limit memory use. These are examples of how we simulated heterogeneous IoT devices.

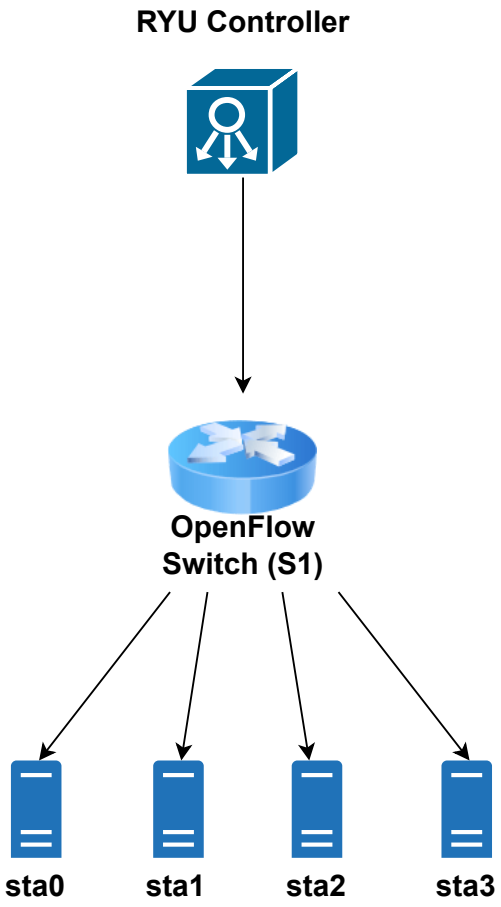

Figure 4.4 Four Client Topology

In Experiments 1–3, a single run of the experiment had multiple sessions; each session contained different attackers and victims. Experiment 4 comprised five separate experimental runs using the same resource configuration but varying attack scenarios.

Experiment	CPU Shares	Memory Limit	Notes
1	10	50 MB	Single experiment with multiple sessions
2	20	100 MB	Single experiment with multiple sessions
3	50	256 MB	Single experiment with multiple sessions
4	10	512 MB	Five independent experiments with different attackers and victims

Table 4.2 Client Resource Constraints Across Experiments

4.2.3 Experiment Duration

The experiment duration was chosen to ensure adequate time for the system to be in normal operational mode and to execute attacks to collect sufficient data for analysis. The average time for Experiments 1-3, which included multiple sessions within an experiment, was 48 minutes per experiment. The average time for Experiment 4, which included separate runs, was 30-35 minutes per run. These durations were

chosen to capture representative system and network information under different resource constraints and attack scenarios, to construct datasets for subsequent training and evaluation.

4.3 Data Collection Methodology

This section describes the methodology used to collect, label, and organize the dataset employed for offline training and validation of the proposed FCDA. The data collection methodology was based on collecting both normal and malicious behavior in SDN-based IoT environments, while maintaining local and temporal consistency for each client.

4.3.1 Attack Scenarios

Three types of volumetric network attacks were selected for generating representative malicious behavior in order to simulate DDoS type attacks: UDP flood, SYN flood, and ICMP flood; these types of attacks are chosen because they have been shown to be prevalent in many DDoS type attacks, and can cause both network and resource usage on hosts to be stressed.

The attacks were generated using `hping3` [20], and executed from within the MininetFed hosts. To increase traffic volume and variability, attack traffic was generated using various `hping3` options, such as `--flood` and `--rand-source`. The roles of attackers and victims were alternated across the four emulated hosts, and the role of each client pair was changed in each experimental session to avoid bias toward any particular client pair.

Each attack session consisted of two intervals. First, an interval of normal behavior occurred prior to the execution of the attack session, and second, an interval of cooldown behavior occurred immediately after the conclusion of each attack session. The fixed duration of each attack session was three minutes, allowing sufficient time for each session to be clearly distinguished as either normal or malicious behavior, and enabling reliable labeling of all collected data during each experiment. Each experiment included multiple attack sessions; the roles of attackers and victims were alternated in each session, and different attacker-victim combinations were used in each session.

Attack Type	Tool Used	Execution Scope	Duration
UDP Flood	hping3	Host-to-host	3 minutes
SYN Flood	hping3	Host-to-host	3 minutes
ICMP Flood	hping3	Host-to-host	3 minutes

Table 4.3 Attack Types and Characteristics.

Despite being effective at generating a stress on both network traffic and host-level consumption of resources, these attack scenarios are restricted to monotonically increasing, network-layer attacks of fixed duration. This choice of attack type simplifies

the experimental setup and makes it trivial to label the resulting dataset, but misses a number of contemporary DoS attacks.

To mitigate this limitation, we report the experiment results in the evaluation phase by attack type. In Section 4.6.3 we discuss detection behavior and errors specific to UDP, SYN, and ICMP floods, and draw attention to differences in Recall and false-negative rates in different attacks. In this way, detection performance is not merely evaluated in aggregate, but with respect to specific attack-types.

More advanced attack classes, such as HTTP floods, Slowloris attacks, DNS amplification, and other application-layer or low-rate attacks are not examined within the scope of this work. Additional protocol-aware and application-level features are needed because these attacks usually show more subtle traffic patterns and sustained resource exhaustion effects. Extending the proposed framework to allow such attacks, as well as including variable and adaptive attack durations, is a key direction for future work.

4.3.2 Feature Collection Methods

Client-side independent data collection occurred across all client hosts using both hardware and network-based methods. Collecting data in this decentralized manner is based on the fact that real-world IoT environments may have limitations, such as no central location to gather data from all clients.

4.3.2.1 Hardware-Based Features

The following hardware-based features were gathered by accessing the `Docker API`. In addition to directly gathering information about the utilization of the resources available to containers, the `Docker API` also provides an opportunity to observe the behavior of processes running within those containers. The collected hardware-based metrics are summarized in Table 4.4.

Category	Metric Name
CPU Utilization	cpu_percent
Memory Utilization	memory_percent
	memory_usage_mb
	memory_limit_mb
Process Activity	process_count
	thread_count
System Load	load_avg_1min

Table 4.4 Hardware-Based Metrics Collected via the Docker API

The hardware-based features provide measures of the computational load experienced by a device, the pressure it experiences due to memory, and the number of active processes at a given time. These are among the most important features for identifying whether a resource-exhausting attack has occurred.

Hardware metrics were sampled at a fixed interval of 3 seconds, providing a balance between temporal resolution and monitoring overhead.

4.3.2.2 Network-Based Features

`Scapy` [5] was used to create network-based features. Using `Scapy` enabled conducting passive packet inspections at the host level. All clients collected and processed their own traffic. The network-based features created from the collected traffic are summarized in Table 4.5.

Category	Feature Name
Traffic Volume	total_packets
	total_bytes
Flow Diversity	unique_src
	unique_dst
Protocol Distribution	proto_tcp
	proto_udp
	proto_icmp
TCP Behavior	syn_count

Table 4.5 Network-Based Features Extracted from Collected Traffic

Network features were aggregated over the same 3-second interval used for hardware metrics. This ensured temporal alignment between system-level and network-level observations.

4.3.3 Dataset Labeling Strategy

To ensure accurate labeling, a time-based labeling methodology was used. For each attack session, a timeline document is produced with two entries: one denoting the beginning of the attack and one denoting its end. Each entry contains a timestamp, the attacker, the victim, the type of attack, and the attack rate.

Sample labels are determined by the temporal overlap between collected sample timestamps and attack interval definitions found in the timeline document. Sample timestamps that fall inside of the attack interval are classified as malicious (1), whereas all other samples are classified as normal (0).

Hardware and network metrics were first collected together in a single CSV file per client. These files are labeled and merged with the timeline file using timestamp alignment to ensure that the features and labels are synchronized correctly.

4.3.4 Dataset Organization

During each experiment, each client host independently generates its own dataset file. The dataset files contain a client_id field (e.g., sta0, sta1, sta2, sta3) and a timestamp to allow for preservation of client-level identity. After each experiment, the labeled per-client datasets are merged into a single dataset containing the experiment's results. In addition, datasets from all experiments are combined into a single, consolidated dataset.

All of the collected dataset files are stored in CSV format. The organization of the dataset allows for per-client analysis to be conducted, while allowing for centralized

offline processing for the purposes of training and validating client models. The dataset's organization enables per-client analysis and centralized offline processing for training and validating client models.

The collected dataset was used exclusively for offline training and validation. A portion of the dataset (20%) was reserved as global test data, later used during online evaluation to measure detection performance in terms of Accuracy, Precision, Recall, and F1-score across client models. The remaining data was used for offline model training and threshold estimation.

Table 4.6 summarizes the distribution of normal and attack samples across individual client datasets.

Client ID	Normal Samples	Attack Samples	SYN_FLOOD	UDP_FLOOD	ICMP_FLOOD
sta0	2732	1074	156	617	301
sta1	2817	1098	484	614	0
sta2	2376	1053	818	235	0
sta3	2804	1130	828	0	302

Table 4.6 Distribution of Normal and Attack Samples Across Clients

4.4 Offline Dataset Preparation and Feature Selection

This section provides an overview of preparing the offline dataset and selecting features for the anomaly detection model's training input. The objective of this stage is to produce a set of features that are informative, non-redundant and compact to support effective unsupervised learning and are deployable in resource constrained IoT environments.

4.4.1 Dataset Cleaning and Normalization

Following data collection and label assignment, the offline dataset was processed through several steps to prepare it for use with the anomaly detection model. Samples in the dataset with either missing values `NaN` or infinite values were removed. Infinite values were initially converted to missing values and then discarded as they could cause numerical instability issues with the training process.

Feature normalization was performed using Min–Max scaling, mapping all feature values into the range [0,1]. The scaler was fitted to the entire dataset collected from all clients, allowing the same normalization scheme to be applied to each client's data when deploying the model. This prevents differences in how different clients scale their feature data from affecting the parameters of the model.

After normalization step, the dataset was divided into separate datasets for each client using the `client_id` attribute. This maintained the ability to access client specific data, but also provided a global representation of feature space for each client.

4.4.2 Training Data Composition

The anomaly detection model was trained only using normal samples, which follows common practices for reconstruction based anomaly detection models. All samples labeled as attacks were excluded from the training phase. This allows the autoencoder to train on a compact representation of normal system behavior without being influenced by abnormal patterns represented in the attack samples.

All attack samples were reserved for validation and evaluation purposes. Evaluating the anomaly detection model was done using a mixed dataset with both normal and attack samples to determine detection performance and provide a basis for establishing the threshold for detecting anomalies. This separation ensures that model training remains unbiased while still enabling quantitative evaluation.

4.4.3 Dataset-Level Analysis of Hardware–Network Relationships

Although the previous exploratory analysis provided an explanation for why the proposed detection framework includes host level hardware metrics, this section will provide a formal data-driven analysis of the relationship between network traffic (i.e., activity) and hardware resource usage. In particular, this analysis is performed on the preprocessed and feature-selected subset of the final offline dataset. This is done to demonstrate that the selected hardware and network features have consistent, measurable and quantifiable relationships to each other while under attack.

4.4.3.1 Temporal Analysis of Hardware and Network Metrics

Figure 4.5 displays a temporal representation of selected hardware and network metrics using a relative time reference. The Figure presents CPU utilization, memory utilization, and SYN packet count over time, with areas of time corresponding to activity due to attacks highlighted. When operating normally, CPU usage is very close to the baseline level, memory usage does not vary significantly, and the number of SYN packets received is very low. These observations are indicative of benign traffic patterns and idle host behavior.

Upon commencement of attack activity, a marked variation is seen in both domains. The sharp increase in the number of SYN packets received indicates the start of a network-based flood attack. Concurrently, CPU utilization rises significantly above normal levels, with the levels of CPU usage varying at substantially higher levels than those observed when no attack is occurring. The increased CPU usage is reflective of the increased processing load that occurs as a result of the high volume of incoming traffic. The memory usage also increases during periods of attack and remains at an elevated level even after the volume of incoming traffic returns to normal, which suggests a persistent strain on system resources.

It should be noted that once the attack has ceased, network metrics return rapidly to baseline levels, while hardware metrics exhibit a more gradual recovery. This difference in recovery rates provides insight into the persistent nature of hardware-related effects beyond the initial period of time when the network-based attack occurred. This type of behavior is particularly relevant to IoT devices, as prolonged resource stress can result in decreased quality-of-service or unstable operation.

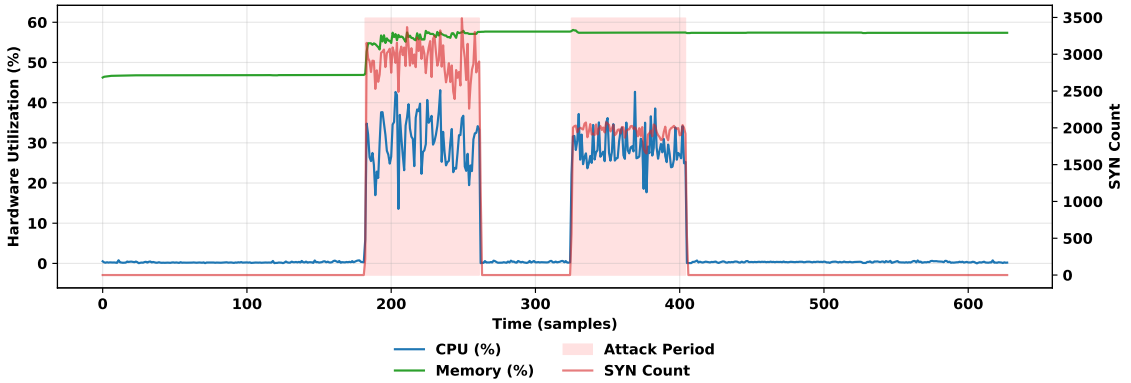

Figure 4.5 Hardware and Network Behavior Over Time (Relative Time)

4.4.3.2 Correlation Between Hardware and Network Metrics

To further quantify the relationship between the usage of selected hardware resources and the activity of network traffic, Figure 4.6 depicts a correlation heatmap computed accross selected hardware and network metrics. The results indicate strong positive correlations between CPU usage and network traffic metrics such as total number of packets and number of SYN packets. The load average metric also exhibits a strong positive correlation with the intensity of network traffic, which is indicative of the cumulative effect of continuous processing of network traffic on system workload.

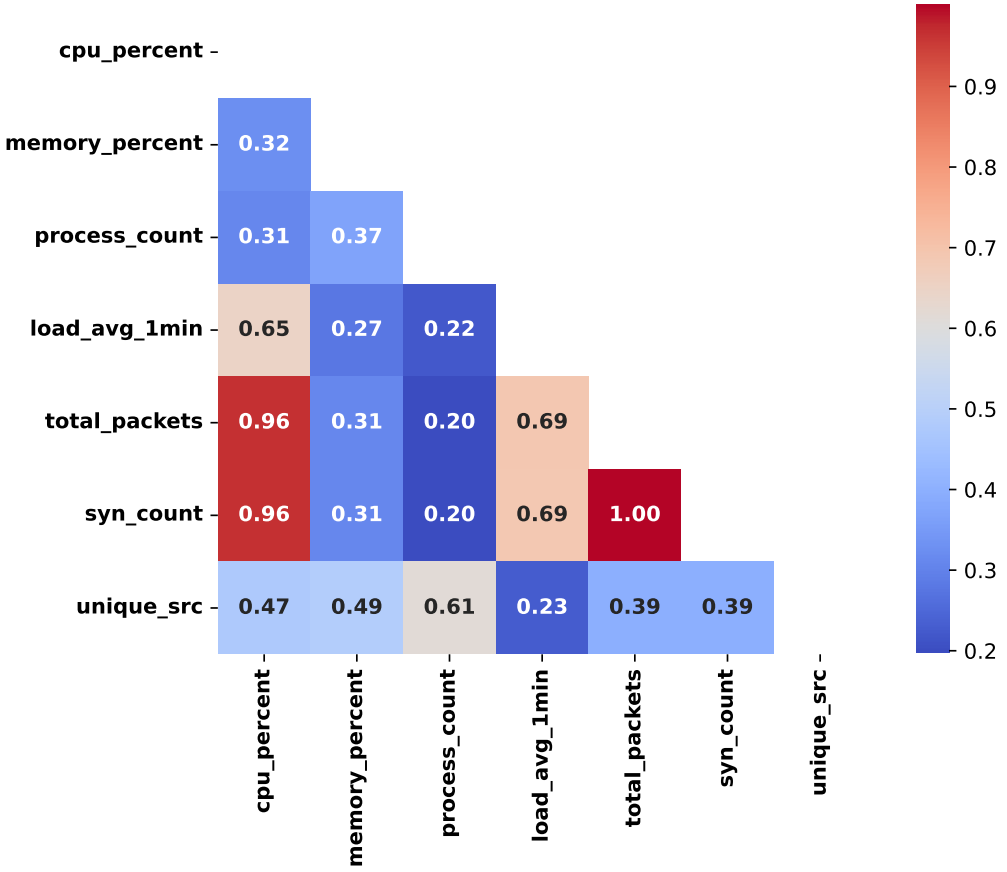

Figure 4.6 Correlation Between Hardware and Network Metrics

Moderate correlations exist between memory usage and network metrics, which suggest that although memory usage increases during attack scenarios, memory usage can be affected by other system factors including buffering and process lifecycle management. A moderate correlation exists between process count and the diversity of sources and destinations, which suggests that traffic patterns resulting from attacks can indirectly affect the processes active on the host.

These correlations quantitatively validate that network-based attacks do not simply result in abnormal traffic patterns but also create measurable stress on host-level resources. These correlations substantiate the incorporation of hardware metrics along with network metrics in the proposed FCDA framework and provide empirical support for the hypothesis that concurrent monitoring improves visibility of anomalies, particularly in resource-constrained environments.

4.4.3.3 Implications for Joint Monitoring in IoT Environments

The findings of this section demonstrate that network-based attacks create simultaneous deviations in both the network and hardware domains. Therefore, reliance on solely network features could potentially overlook early or subtle signs of attack-induced resource stress, particularly in IoT deployments where even slight increases in CPU or memory usage can be critical. Incorporating metrics related to usage of available host-level resources into the detection pipeline enables the proposed FCDA framework to capture a broader operational context and therefore enables the development of a more robust and resilient anomaly detection capability.

This empirical study provides the foundation for the feature selection process discussed in subsequent sections and provides justification for the decision to combine hardware and network metrics within a FL based detection architecture.

4.4.4 Feature Redundancy Analysis

Prior to reducing the number of features, a Pearson correlation analysis was applied to the entire set of collected features. A correlation heat map of this analysis is presented in Figure 4.7.

As illustrated in Figure 4.7, there were many features that had significant linear correlation to one another. More specifically, features that represent network traffic volume metrics as well as those representing process metrics correlated very strongly. There were also strong correlations between certain packet level volume features as well as between process and thread count features. These strong correlations suggest that there is redundancy in these features, which could result in retaining all of them in the model and therefore increase its complexity without providing significantly more information.

Instead of setting a predetermined correlation threshold to eliminate redundant features, the results of the correlation analysis were used to assist with selecting features based on their relationship to other features.

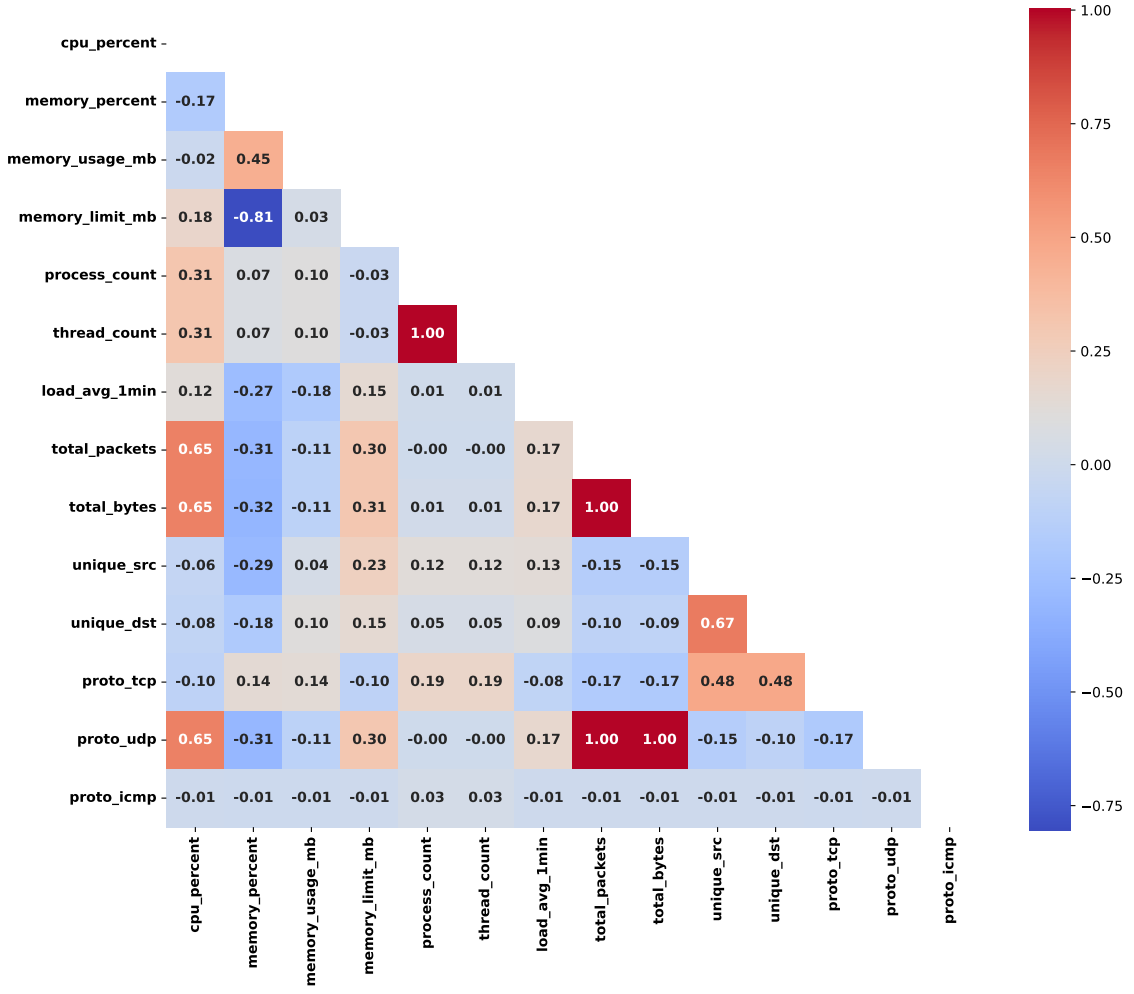

Figure 4.7 Feature Correlation Heatmap

4.4.5 Feature Relevance Assessment Using Mutual Information

Mutual information (MI) was calculated between the candidate features and class labels to assess the relevance of each feature to detecting attacks. MI captures both linear and nonlinear relationships between the target variable and each feature. The MI values for all candidate features are shown in Figure 4.8.

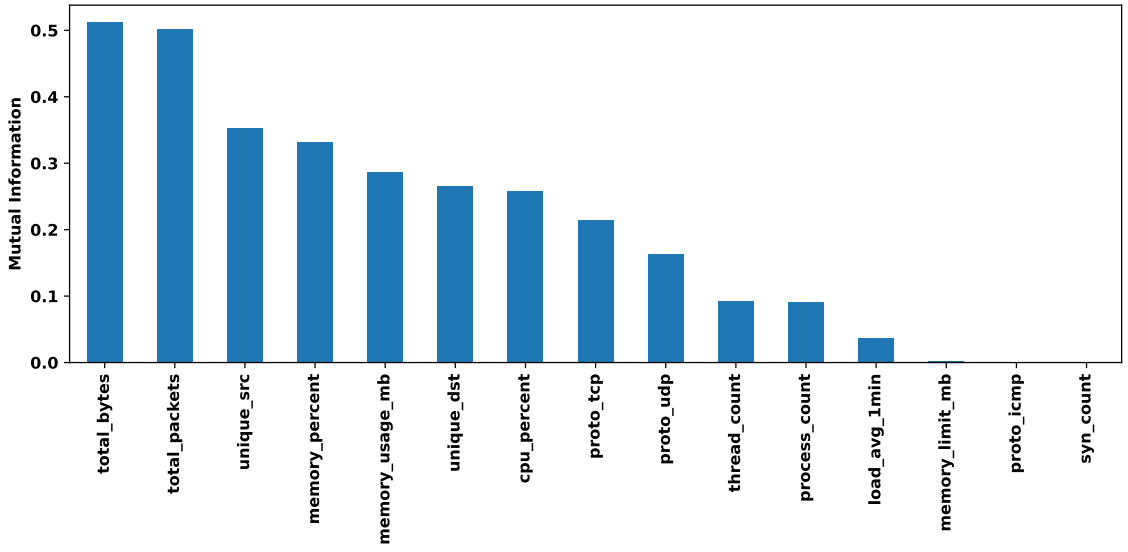

Figure 4.8 Mutual Information Scores of Features

Results from the MI analysis shown in Figure 4.8 demonstrate that many of the network traffic intensity features along with some of the selected hardware metrics provided very strong discriminatory information about normal and attack behavior. On the other hand, some of the candidate features have MI values that are close to zero, indicating that they do not contribute much to improving detection performance.

The MI analysis provides further insight into the correlation analysis by showing that some of the most informative features can be relevant to the task, even though there may be redundancy between some of the features. Therefore, this analysis supports a balanced approach to feature selection that takes into account both the relevance of the features as well as redundancy.

4.4.6 Final Feature Selection

The results of correlation analysis, MI analysis, and the model-based experiment eliminating redundant features, indicate an optimal subset of ten features that can be used to support the detection model's ability to identify hardware anomalies through their relationship with network-related activity.

Evidence driven manual selection was employed for the identification of this subset of features, and priority was given to those features demonstrating the highest degree of relevancy and to eliminate or minimize any potential for redundant information and high computational costs.

The final selected feature set used for model training is summarized in Table 4.7.

Category	Feature Name
System Resource	cpu_percent
	memory_percent
	process_count
	load_avg_1min
Network Traffic	total_packets
	unique_src
	unique_dst
	proto_tcp
	proto_icmp
	syn_count

Table 4.7 Final Selected Feature Set

These selected features provide a broad range of related, but complementary views of system level resources and network-related activities; thereby providing the model with the means to effectively relate hardware anomalies to network-related anomalies.

4.4.7 Offline Dataset Split

Preparation of the dataset for the purposes of training and evaluating the model involved partitioning it into three separate partitions: training, validation and test. Normal sample data was utilized solely for the purpose of training the autoencoder and estimating reconstruction-based detection thresholds. A mixed dataset containing both normal and attack samples was randomly split, with 20% reserved as a global test set.

The random partitioning of the dataset was done without setting a fixed global seed to prevent bias within the dataset partition, and to simulate the variability found in different deployment environments. The global test set would later be used to calculate various metrics associated with detection performance of each client model during online evaluation phase (e.g., Accuracy, Precision, Recall, and F1-score).

4.4.8 Summary

Data pre-processing and feature selection, using the tools provided by correlation and MI analyses with empirical validation resulted in a compact, relevant representation of the dataset which supports unsupervised anomaly detection. This dataset will serve as the foundation for offline model training and for the creation of the deployment artifacts discussed in subsequent chapters.

4.5 Online Deployment and Experimental Procedure

This section provides information about the experimental setup with respect to the live MininetFed environment and the deployment of the proposed FCDA. The

objective of this stage is to analyze the operational properties of the proposed system under continuous monitoring, detection, and federated adaptation.

4.5.1 Online Deployment Configuration

To ensure that all clients started from the same initial state in a federated system, each client agent was configured with a set of pre-computed, offline-generated deployment artifacts. These artifacts included the autoencoder model weights from offline training, the parameter values of the Min–Max scaler used for feature normalization, and a set of client-specific hyperparameters (i.e., the learning rate, the number of local epochs per iteration, and the anomaly detection threshold).

Although all clients used the same autoencoder architecture and initial model weights, they employed client-specific parameters at runtime. This design approach permitted a common representation of normal behavior while accommodating heterogeneous learning dynamics across clients.

The deployment artifacts files were loaded into each client agent during the client initialization process, at which point each client entered into a continuous monitoring and detection cycle. No design changes were made between the offline and online systems, ensuring an accurate and consistent representation across the two stages.

4.5.1.1 Anomaly Detection Threshold Selection and Sensitivity Analysis

Anomaly-detection threshold values play a significant role in reconstruction-error-based anomaly detection methods, as they determine the trade-off between false positives and false negatives. For the proposed framework, the anomaly detection threshold for each client was determined during the offline knowledge-generation phase using labeled training data and remained constant during online deployment. This method provides consistent detection behavior without drifting in the threshold during runtime.

A post-deployment sensitivity analysis was performed using the reserved global test dataset to support the selection of the threshold values. The analysis evaluated detection performance across a range of threshold values by calculating the mean Precision, Recall, and F1-score across all clients. Figure 4.9 illustrates the relationship between the anomaly threshold and these three metrics.

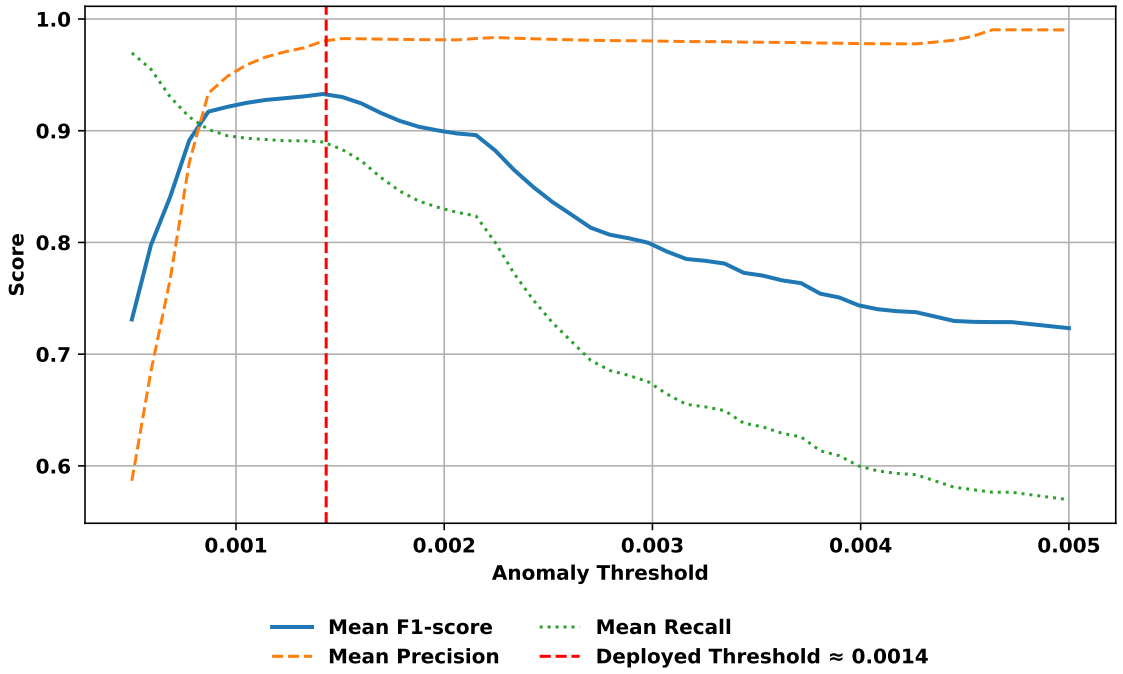

Figure 4.9 Threshold Sensitivity Analysis

The results demonstrated that the system's precision increased monotonically with increasing threshold values, while Recall decreased with decreasing numbers of anomalous samples exceeding the threshold. The mean F1-score reached its maximum in a region of stability about the deployed threshold value (approximately 0.0014), and it was important to note that this region exhibited a plateau rather than a sharp peak, indicating that detection performance is robust to small threshold variations.

The threshold value used for the online deployment of the client agents was deliberately selected within the stable operating region to balance detection effectiveness and operational reliability. Lower threshold values improve Recall at the expense of a higher false-positive rate. False positives can be particularly detrimental in resource-constrained IoT environments, as they can lead to unnecessary mitigations that disrupt the system's normal operation. On the other hand, higher threshold values yield fewer false positives but fewer detections for low-intensity or short-duration attacks.

This behavior matches the confusion matrix and attack type error analysis presented in Section 4.6.3, where false positives are few and false negatives are primarily related to low intensity UDP flood traffic. Therefore, the threshold sensitivity study demonstrated that the threshold values obtained from offline training are justified and suitable for online federated deployment.

4.5.2 Online Monitoring and Detection Loop

During each cycle of continuous monitoring and detection, each client agent collected real-time data concerning both hardware and network metrics. At predefined intervals, the client would extract a feature vector from the collected data and apply the preloaded scaler parameters to normalize it.

Anomaly detection was then performed by calculating the reconstruction error obtained when the normalized feature vector was input to the client's autoencoder. When the calculated reconstruction error exceeded the client-specific threshold determined through offline validation, the client would classify the input feature vector as anomalous. Anomalies were identified on a per-sample basis, thereby permitting early detection of abnormal behavior.

In addition to normal operation, controlled DDoS attack traffic was periodically generated during the experiment using `hping3` between selected host pairs. These DDoS attacks served only to validate the client's response during live operation and were not used as part of the performance evaluation metrics.

4.5.3 Federated Training Protocol

The FL process was implemented in an online experiment over predetermined number of rounds. There was no pre-specified termination point, the system was intended to run continuously; the experiment was stopped manually once the predetermined number of rounds had been completed.

All clients were eligible to take part in FL in each round. Eligibility to perform local training was conditional upon having enough confident normal samples available. Therefore, clients would perform local updates only if they met this criterion; otherwise, they would submit an empty update, which would be excluded from the aggregation for that round.

Local training was executed according to each client's specific learning rate and epoch configuration, as provided in the deployment artifacts. This approach prevented overfitting to limited amounts of noisy data while minimizing the additional processing load that may be incurred on resource-constrained devices from unnecessary computation.

4.5.4 Server-Side Aggregation and Model Distribution

FedAvgM is used for aggregating the client updates to create the global model. FedAvgM is similar to standard federated averaging except FedAvgM includes a momentum term to improve stability in the presence of non-identical data distributions among clients.

Updates were aggregated on the server at the completion of each federated round and included only those from clients that contributed a valid local model. The resulting global model was then broadcast to all clients, ensuring each maintained a consistent version, regardless of whether they participated in the most recent round.

Continuous updating of the global model was achieved while simultaneously enabling robustness against client heterogeneity and variability in client participation.

4.5.5 Experimental Procedure Summary

The online evaluation followed a structured and repeatable experimental workflow. The first step involved the distribution of deployment artifacts produced during the

Parameter	Description
Initial weights	Loaded from offline training
Scaler	Min–Max (offline fitted)
Threshold	Client-specific
Aggregation	FedAvgM

Table 4.8 Online Deployment Parameters

offline knowledge generation phase to all client agents. Subsequently, a continuous monitoring and anomaly detection cycle began for each client agent. During this cycle, local feature vectors were processed by the client and reconstruction errors calculated using the autoencoder model; once there were sufficient normal observation values, conditional local training was conducted at the client level. Local model updates were periodically transmitted to the federated server, where aggregation was carried out using the FedAvgM algorithm. The newly created global model would then be returned to all the client agents and the process would continue to repeat continuously during runtime operation.

To avoid a single execution being the source of the results presented, the complete online experiment was executed independently three times, each time consisting of approximately 800 federated rounds. The same deployment artifacts, topology configuration and attack scenarios were utilized in each of the three runs. To evaluate the detection performance metrics (i.e., Accuracy, Precision, Recall and F1-score), at each federated round, the previously reserved global test dataset was used. Thus, the short-term learning dynamics and the long-term stability of the proposed FCDA system could be evaluated.

Table 4.9 reports the final detection performance for each independent run, together with the mean and standard deviation across all three runs. The results show that the proposed system has low variance across the multiple executions, and that the Accuracy and Precision remain stable with the standard deviation of the metrics remaining small (on the order of 10^{-3}). These results indicate that the detection performance is reproducible and robust to the variability of the runtime environment.

Run	**Accuracy**	**Precision**	**Recall**	**F1-Score**
1	0.957	0.971	0.880	0.923
2	0.953	0.967	0.870	0.916
3	0.957	0.973	0.878	0.923
Mean ± Std	0.956 ± 0.002	0.970 ± 0.003	0.876 ± 0.005	0.921 ± 0.004

Table 4.9 Final Detection Performance Across Independent Federated Runs

To further investigate the convergence characteristics of the proposed system, Figures 4.10 and 4.11 show how the average detection accuracy and global reconstruction loss evolved over federated rounds for the all three runs. From Figure 4.10, it can be seen that the Accuracy reaches a stable point early in the experiment and then only marginally varies in the subsequent rounds, with most of the variations being less than one-third decimal place. Furthermore, from Figure 4.11, it is clear that the global reconstruction loss remained bounded within a relatively narrow range across all runs, without signs of divergence or oscillatory behavior. The temporary

deviations observed in the reconstruction loss in some of the iterations were found to be self-correcting and did not cause any long term decline in the overall performance of the system.

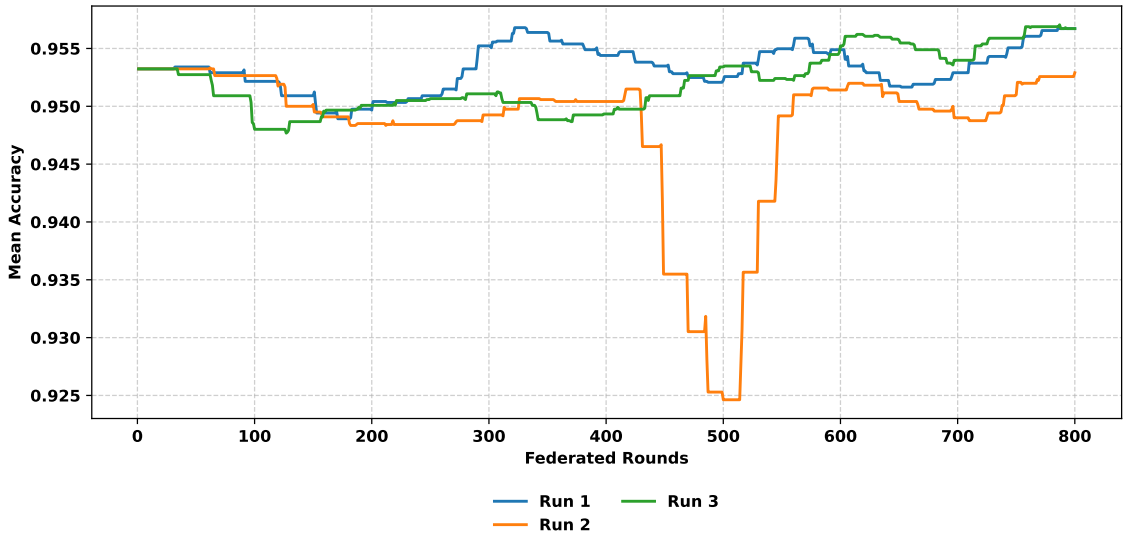

Figure 4.10 Mean Accuracy vs Federated Rounds

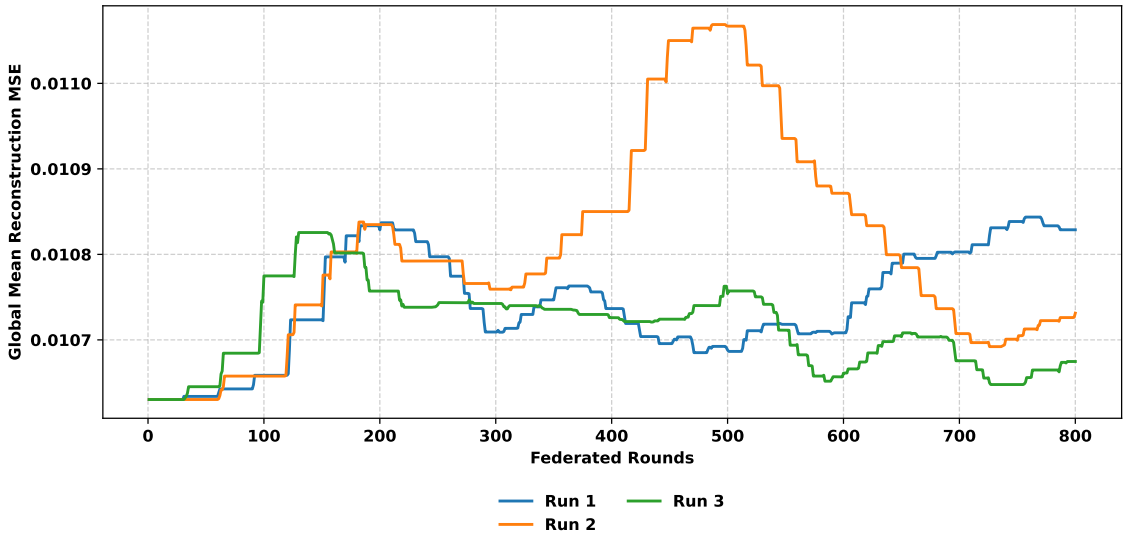

Figure 4.11 Global Mean Reconstruction MSE

Therefore, these results confirm that the proposed FCDA framework converges reliably and demonstrates stable detection performance during extended periods of operation. Unlike other systems, the proposed system is designed to operate continuously and reflects realistic deployment scenarios in SDN-based IoT environments where monitoring and adaptation occur indefinitely. The convergence characteristics observed in the experiments justify the duration of the experiments and confirm the reliability and stability of the FL process in accordance with the design of the proposed system.

4.6 Evaluation Metrics and Results

This section presents the quantitative assessment of the FCDA. This includes an assessment of detection capabilities, robustness across the federation of clients and how use of offline knowledge impacts the overall effectiveness of the system. All evaluation metrics presented are based on the reserved global test set (20%) which contains both normal and attack samples. Further, it is separate from the data used to train the models.

Unless noted otherwise, all of the details provided in sections 4.6.1–4.6.4 regarding the detection performance have been generated as part of a single experiment run over 1200 federated rounds. These rounds were selected to represent realistic continuous operations of the architecture, allowing for a complete analysis of the detection patterns, including the confusion matrices, the type of errors made for each type of attack; ROC curve analysis. In addition to the above referenced evaluation run, independent evaluation runs were performed and documented separately in Section 4.5.5. These independent evaluation runs assessed the overall robustness of the FL approach, its stability, and its convergence by reporting mean performance and their corresponding variances across executions.

4.6.1 Evaluation Metrics

Binary classification metrics are employed to evaluate detection performance:

- **Accuracy**: Measures the overall correctness of predictions.
- **Precision**: Quantifies the proportion of true positive detections out of total detections.
- **Recall**: Measures the proportion of true positive detections out of all possible positive detections.
- **F1-Score**: Represents the harmonic mean of Precision and Recall.

Metrics are calculated per client at the end of the evaluation rounds and then averaged across all clients to yield mean performance metrics. There is no temporal smoothing; therefore, the metrics reflect the model's instantaneous performance at its final state.

4.6.2 Detection Performance Across Clients

Results of the final detection capability of the offline initialized FCDA are summarized for each client in Table 4.10. These results show consistent Accuracy and Precision across all clients, despite non-identical data distributions and diverse local observations. This similarity in clients' performance can be attributed to the fact that the offline knowledge-generation process successfully produced a robust initialization that performed adequately across all clients, regardless of the differing traffic and resource conditions each client operated under.

Client	Accuracy ± 95% CI	Precision ± 95% CI	Recall ± 95% CI	F1-Score ± 95% CI
sta0	0.960 ± 0.007	0.982 ± 0.009	0.880 ± 0.022	0.928 ± 0.012
sta1	0.961 ± 0.007	0.974 ± 0.010	0.891 ± 0.019	0.931 ± 0.013
sta2	0.960 ± 0.007	0.982 ± 0.008	0.880 ± 0.022	0.928 ± 0.012
sta3	0.961 ± 0.007	0.974 ± 0.010	0.891 ± 0.019	0.931 ± 0.012

Table 4.10 Final Detection Performance per Client (Offline-Initialized FCDA)

In addition to assessing overall performance, Table 4.11 provides information on the stability of the proposed system's performance. Table 4.11 presents the mean detection metrics (i.e., Accuracy, Precision, Recall, F1-Score) for all clients, along with their corresponding 95% confidence intervals. The proposed system demonstrated a mean Accuracy of 0.960 ± 0.001 and a mean Precision of 0.978 ± 0.005, indicating high classification Accuracy and strong confidence in attack prediction. The narrow CI of the metrics for Accuracy and Precision indicates that there is little variation in performance among the clients and therefore the FCDA system behaves reliably and predictably in a distributed environment.

The Recall rate of 0.885 ± 0.006 is significantly lower than the Precision rate of 0.978 ± 0.005. The persistent gap between Recall and Precision indicates an inherently conservative anomaly-detection method within the autoencoder-based approach. Although Precision takes precedence over Recall in this approach due to the need to avoid false positives, which could disrupt services or lead to undue resource utilization in IoT networks. However, this conservative behavior may result in some attacks not being detected, as reflected in the lower Recall.

Metric	Mean ± 95% CI
Accuracy	0.960 ± 0.001
Precision	0.978 ± 0.005
Recall	0.885 ± 0.006
F1-Score	0.930 ± 0.001

Table 4.11 Mean Detection Performance Across Clients

A discrepancy between Recall and Precision does not indicate instability; the narrow confidence interval for Recall indicates that the missed-detection events occur systematically rather than randomly, thereby motivating further investigation into the nature of these false negatives. Further investigation into this area is addressed in Section 4.6.3, where confusion matrix analysis and attack-type-specific error breakdowns reveal that the majority of missed detections correspond to specific attack categories rather than generalized model failure.

In summary, the results presented in Tables 4.10 and 4.11 show that the offline-initialized FCDA framework produces stable, high-recision detection across clients, with predictable, explainable trade-offs between Precision and Recall. The characteristics inherent to the proposed methodology provide the basis for its suitability for deployment in resource-constrained SDN-based IoT environments, where reliability and low false positive rates are critical.

4.6.3 Confusion Matrix and Attack-Type Error Analysis

An aggregated confusion matrix was used to better understand how classification occurs. By summing up the amount of true positives, false positives, true negatives, and false negatives over all of the clients in the final evaluation round, a total confusion matrix was created. The resulting confusion matrix is shown in Figure 4.12.

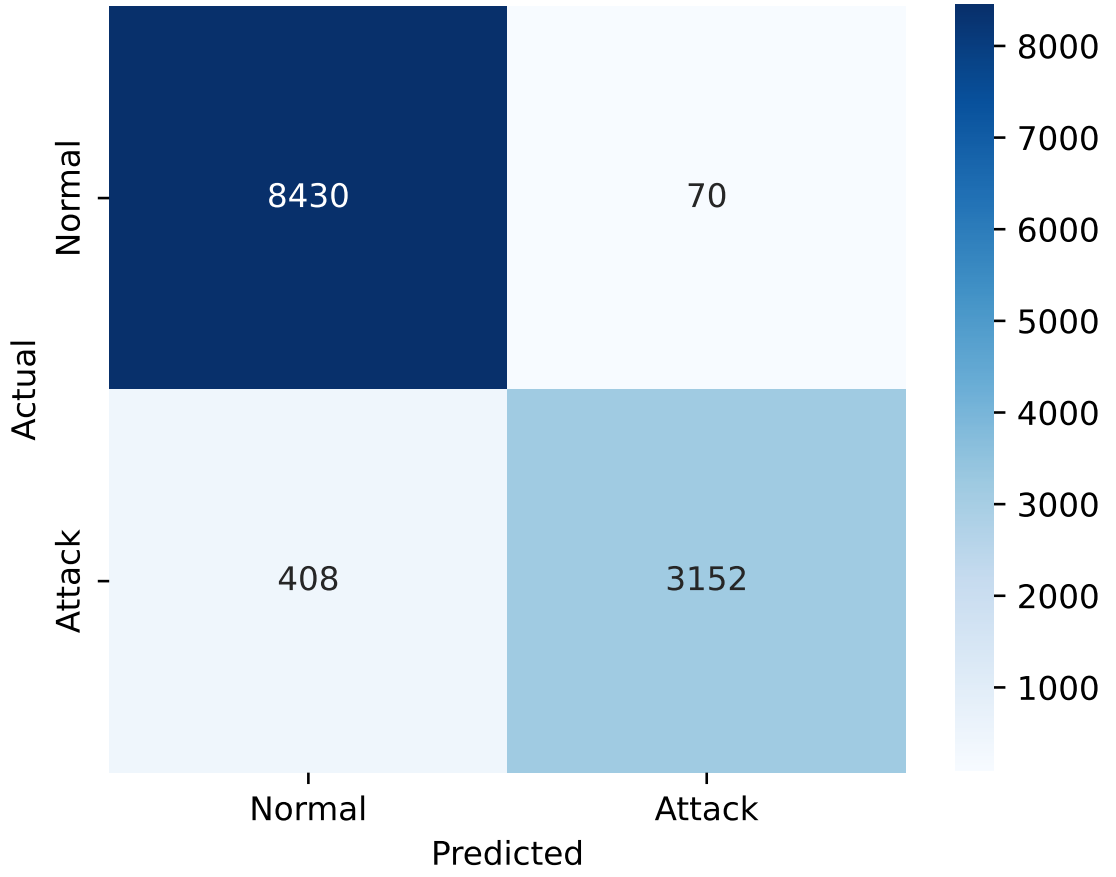

Figure 4.12 Aggregated Confusion Matrix (Offline-Initialized FCDA)

Based on the results from the confusion matrix, it can be seen that the majority of the normal samples were correctly classified and there were very few false positives generated; this demonstrates the ability of the offline derived anomalies thresholds to identify normal traffic patterns. Additionally, while a small number of false negatives exist the overall ratio of Precision to Recall suggests that the FCDA has good ability to detect attacks in real-time monitoring environments.

Although a small amount of false negative classifications but they are not equally distributed among different attack types. A detailed review of false negative classifications by attack type revealed that the vast majority of false negatives resulted from undetected UDP flood attacks. However, both SYN flood and ICMP flood attacks had a much higher Recall rate than UDP floods. This is summarized in Table 4.12, which presents the total number of attack samples, false negatives, and the false negative rate for each attack type. The results indicate that UDP flood

attacks exhibit a substantially higher false negative rate (32.4%) compared to SYN (1.46%) and ICMP (1.61%) floods.

Attack Type	Total Attack Samples	False Negatives	False Negative Rate (%)
UDP_FLOOD	1148	372	32.4
SYN_FLOOD	1916	28	1.46
ICMP_FLOOD	496	8	1.61

Table 4.12 False Negatives Distribution by Attack Type

In addition, the analysis of detection performance by attack type showed that the Precision achieved by the FCDA framework is 1.0 (Table 4.13) for all of the attack categories evaluated. This means that once an attack has been reported, it is always accurate and none of the reported attack samples are incorrectly associated with another class. This demonstrates the extremely conservative nature of the threshold-based autoencoder design of the FCDA framework, where it will only report an anomaly if it has high confidence. However, this conservative behavior results in lower Recall for certain attack types, most notably UDP floods leading to missed detections under specific traffic conditions.

Attack Type	Precision	Recall	F1-Score	False Negatives
UDP_FLOOD	1.0	0.675958	0.806653	372
SYN_FLOOD	1.0	0.985386	0.992639	28
ICMP_FLOOD	1.0	0.983871	0.991870	8

Table 4.13 Detection Performance by Attack Type

A higher number of false negatives were detected for UDP Flood Attacks due to their connectionless nature and the resulting traffic characteristics. Unlike SYN and ICMP floods, which result in sudden and protocol-specific overhead in processing (i.e. connection establishment or control message handling), UDP floods tend to result in uniform and continuous traffic patterns. Such patterns can result in sustained loads on hosts with relatively minor and short-term variations in reconstruction errors. Therefore, low-rate or shorter-duration UDP Flood Attacks may not consistently exceed the fixed anomaly threshold and thus may result in undetected attacks.

This difference in Precision and Recall is reflective of a trade-off in anomaly-based intrusion detection systems. The conservative approach used by the FCDA framework to ensure high Precision, thereby minimizing false alarms and ensuring the stability of the system in resource constrained IoT deployments, also results in a reduced Recall for UDP floods. Such a reduced Recall suggests that certain low-level or short duration attacks may not always have sufficient anomalous characteristics to exceed the anomaly threshold of the system. Implementing dynamic adjustment to the detection thresholds of the system or incorporating attack-specific adaptation mechanisms could improve Recall without significantly increasing false positives and this is identified as an important direction for future work.

Overall the confusion matrix and the attack-type error analysis demonstrated the reliability of the proposed FCDA framework in confirming attacks while providing robust classification of normal traffic, thereby supporting its applicability for real-time deployment in SDN-based IoT environments.

4.6.4 ROC Curve Analysis

Receiver Operating Characteristics (ROC) curves were used to determine how well the autoencoder could differentiate between attack and normal samples through the use of the continuous reconstruction errors (MSE). The ROC curve shows the relationship between the true positive rate and the false positive rate for different values of the threshold. The results are presented in Figure 4.13.

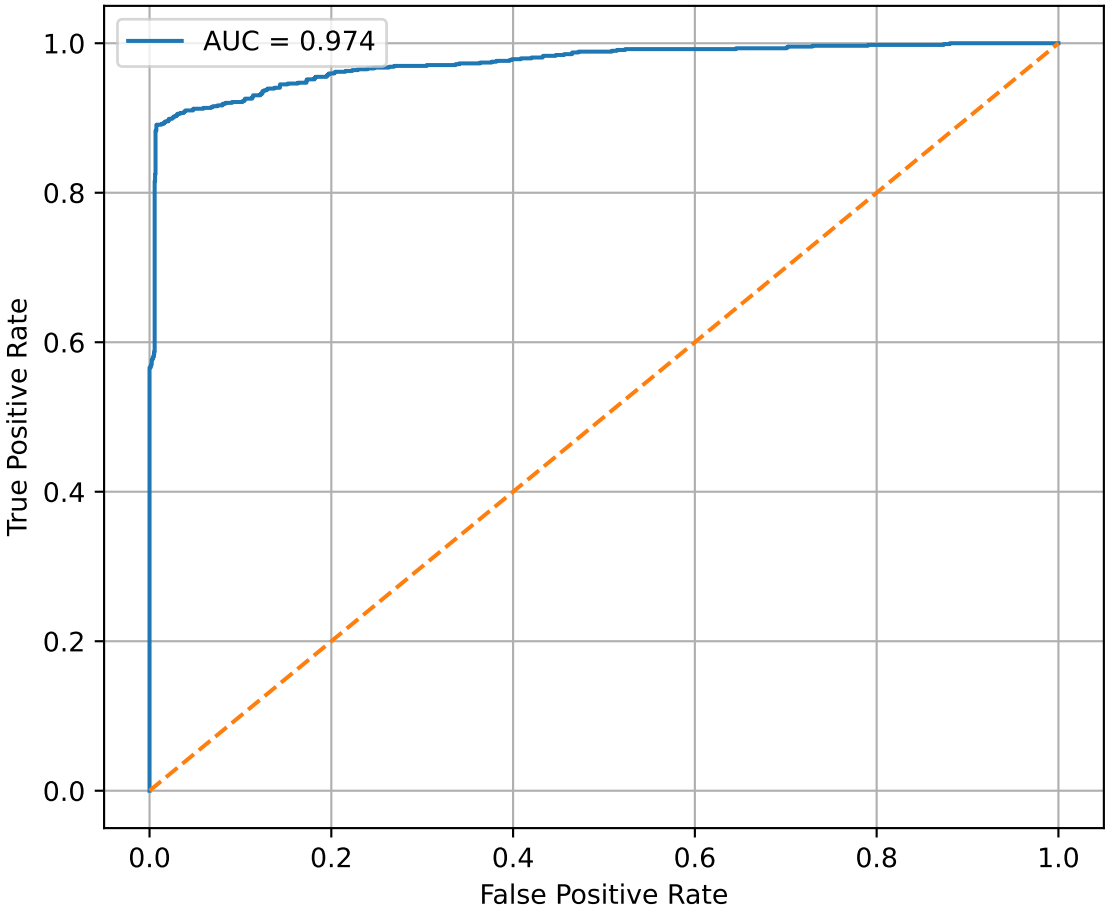

Figure 4.13 ROC Curve (Offline-Initialized FCDA)

Using the area under the ROC curve (AUC), it was determined that the autoencoder learned a discriminatory representation of normal behavior and therefore had a high degree of separability when classifying samples between normal and attacks, with an AUC value of 0.974.

4.6.5 Comparative Evaluation with FL from Scratch

The impact of offline knowledge generation on the proposed FCDA was evaluated using a baseline scenario in which FL was performed from scratch, i.e., without pre-trained model weights or prior knowledge from normalization parameters. This baseline scenario utilized the exact same deployment and evaluation as the proposed FCDA, however it accumulated all its training data while operating.

Figure 4.14 illustrates the overall detection performance of both scenarios. The offline-initialized FCDA demonstrated consistent Accuracy, Precision, and F1-scores, while the baseline showed significantly reduced Accuracy and Precision values and therefore poor anomaly detection capability in the absence of prior knowledge.

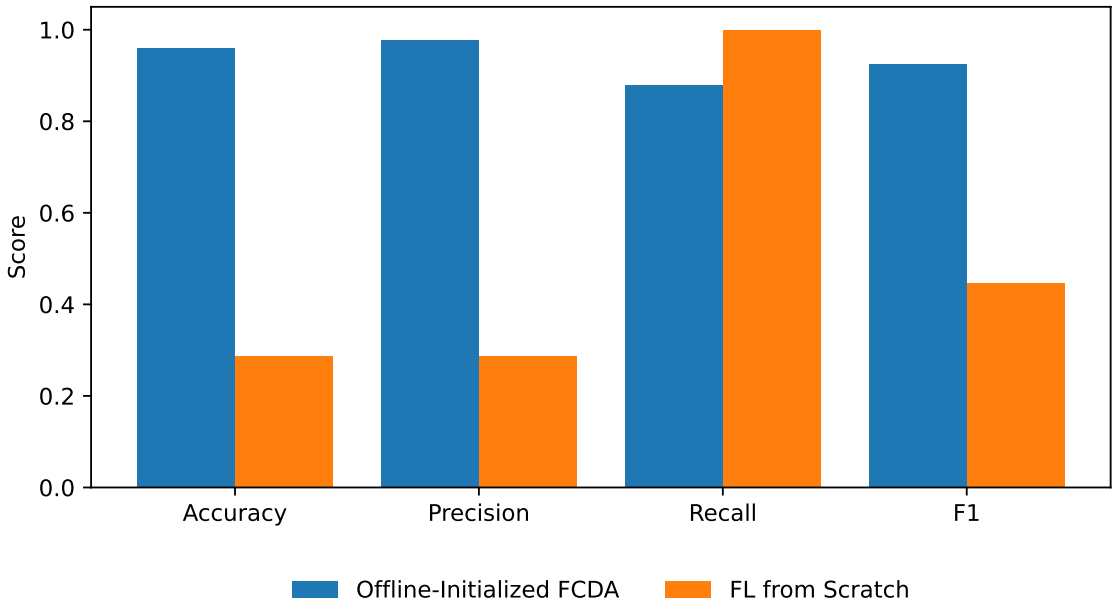

Figure 4.14 Final Detection Performance Comparison

In addition, to better understand the learning process depicted in Figure 4.15 shows the mean detection accuracy over the first 100 federated rounds for both the proposed FCDA and the baseline. For fair comparison between the two experiments, the Accuracy of the offline-initialized FCDA was cut off after 100 rounds to match the duration of the baseline experiment. The results demonstrate that the offline-initialized FCDA, maintained a very high Accuracy value, right from the start, indicating operational readiness immediately. Conversely, the baseline did not exhibit any learning growth and remained at a very low Accuracy value throughout the entire experiment.

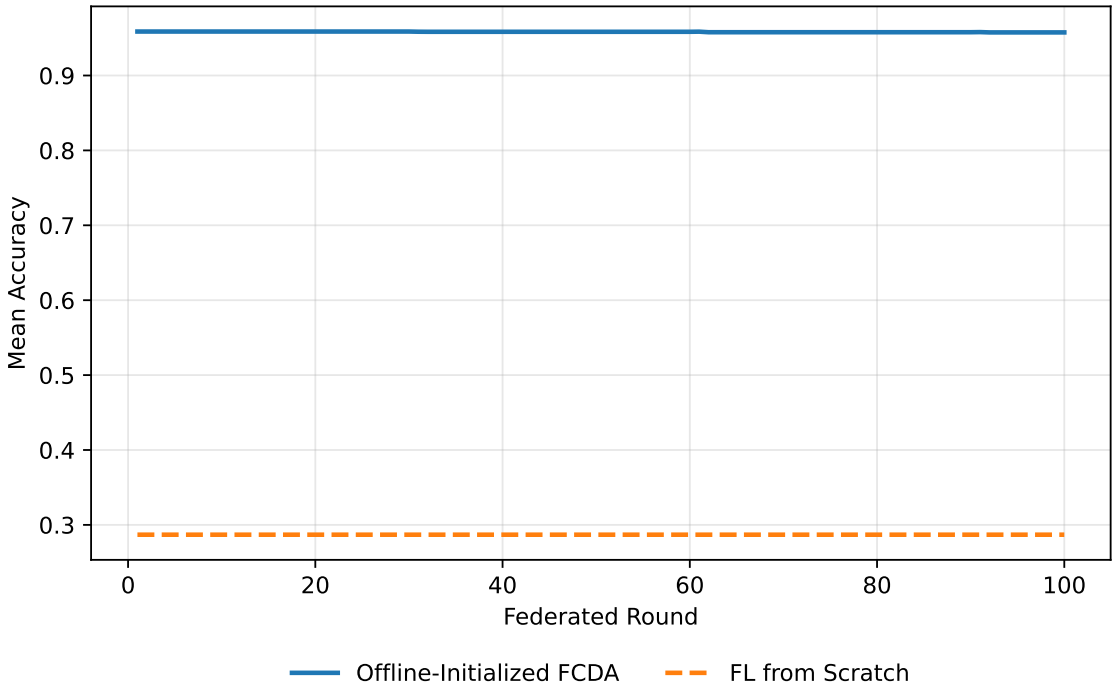

Figure 4.15 Mean Detection Accuracy vs Federated Rounds

This inability to learn by the baseline configuration can be attributed to the absence of an initial model representing normal behavior, combined with limited local data buffering under resource constraints. Therefore, the baseline system was incapable of gathering enough reliable samples for it to provide stable federated updates. Therefore, the baseline system was incapable of gathering enough reliable samples for it to provide stable federated updates.

Overall, this comparative analysis clearly demonstrates that offline knowledge generation is a key component of the proposed FCDA. Through its ability to provide an informed initialization, the offline phase allows for the effective and stable detection of anomalies within federated systems that have constrained resources, which cannot be achieved through FL from scratch alone.

4.7 Scalability Analysis

Section 4.6 demonstrated the feasibility of the proposed FCDA framework in a small-sized federated environment; however, in most cases, a realistic SDN-based IoT deployment includes many more distributed units. Thus, this section will investigate whether the proposed system maintains its ability to detect anomalies accurately as the number of clients increases.

In particular, this section focuses on the scalability of the proposed FCDA framework by investigating the behavior of the proposed architecture when the network size increases. In contrast to creating new offline-trained models for each network size, the proposed experiments evaluate the structural and functional scalability of the proposed architecture, i.e., whether the offline-initialized FL process remains

stable, whether the detection performance deteriorates as the federation expands, and whether the aggregation mechanism still operates correctly with more clients.

Two scalability experiments were conducted utilizing larger networks with 8 and 16 clients, respectively. The detection performance achieved in these experiments are compared to the detection performance of the results reported in Section 4.6, where a network with 4 clients was used.

4.7.1 Experimental Setup for Scalability Analysis

For the purpose of comparison to the 4-client experimental setup presented in Section 4.6, the scalability experiments were conducted utilizing the same system design. Each client utilized the same autoencoder architecture, feature set, anomaly detection logic, and federated aggregation method as before. The same methodologies for generating attacks, and evaluating detection performance, were used as well.

For the scalability experiments, the same deployment artifacts generated offline from the 4-client knowledge generation process were used. These artifacts comprised the trained autoencoder model weights, the Min-Max scaler parameters, and the client-specific hyperparameters. For the additional clients added to the network in the expanded configurations, the deployment parameters were replicated from the original four clients, in a cyclical manner. This method provided for a controlled evaluation of scalability without requiring repeated offline retraining.

Each scalability experiment was executed using MininetFed, and the clients operated within resource constrained container environments similar to the previous experiments. The detection performance for each of the scalability experiments was evaluated using the same reserved global test dataset, so that comparisons of detection performance could be made fairly and consistently, across different network sizes.

The number of rounds utilized in the federated scalability experiments differed from the number of rounds utilized in the prior 4-client experimental evaluation. This choice was influenced by the convergence analysis presented in Section 4.5.5, where it is shown that the FCDA framework reaches stable detection performance and bounded reconstruction loss early during runtime operation with minimal variability in subsequent rounds. Therefore, continued execution past the point of convergence was not required to evaluate scalability. The 8-client and 16-client experiments were therefore terminated after 350 and 300 rounds respectively, which were sufficient to achieve and maintain stable post-convergence behavior while minimizing unnecessary computational overhead. This approach provides a realistic representation of the typical deployment scenario where scalability is evaluated under steady-state operation, rather than prolonged training durations.

4.7.2 Experiment 1: 8-Client Deployment

The first scalability experiment examined the proposed FCDA framework in an 8-client federated deployment. The network configuration utilized one ryu remote

controller and two OpenFlow switches (S1 and S2), which were interconnected. 4 clients were attached to each switch, yielding a total of 8 distributed clients.

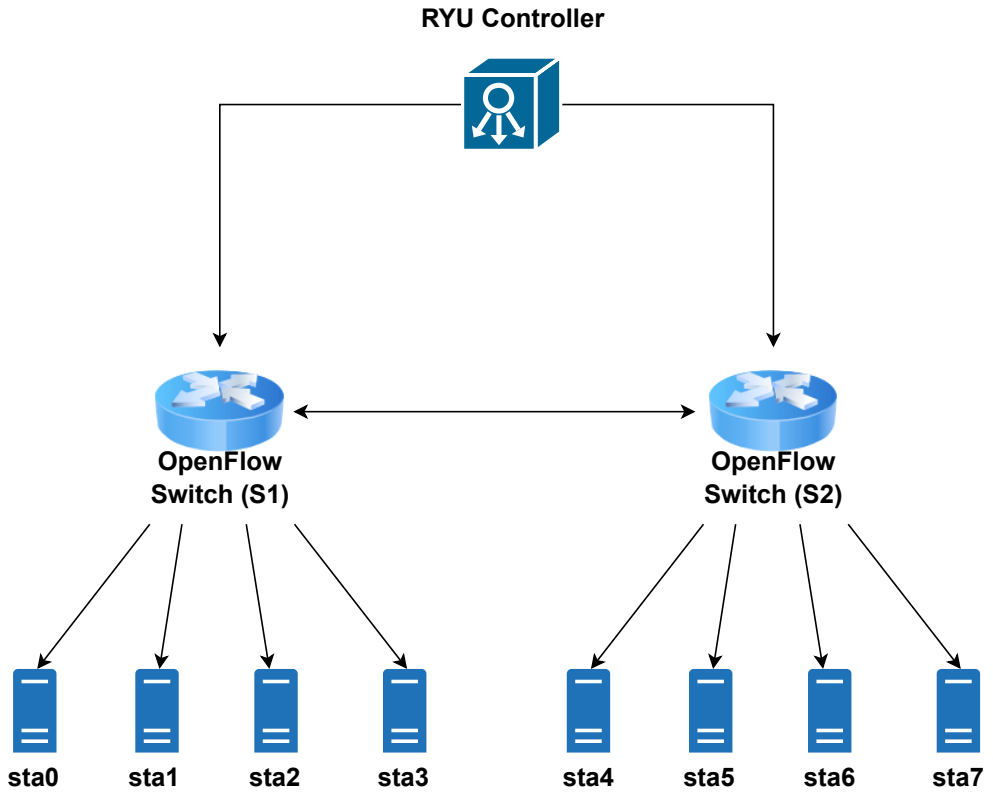

Figure 4.16 Eight Client Topology

The additional clients were configured by replicating the offline generated deployment parameters of the original four clients. Specifically, the deployment parameters of clients sta0 through sta3 were duplicated for clients sta4 through sta7, respectively. In doing so, the federation was scaled, and the use of the validated offline initialization process was maintained.

The experiment was performed over 350 FL rounds. During the experiment, the clients performed continuous monitoring, detected anomalies, and conditionally trained locally as explained in Section 4.5. The federated averaging aggregation was performed using the FedAvgM algorithm, and detection performance metrics were logged per client after each round.

The final-round detection performance across all 8 clients is summarized in Table 4.14. The values provided represent the mean detection performance of the clients at the final evaluation round along with the corresponding 95% confidence interval.

Metric	Mean ± 95% CI
Accuracy	0.956 ± 0.001
Precision	0.968 ± 0.002
Recall	0.880 ± 0.006
F1-Score	0.922 ± 0.003

Table 4.14 Mean Detection Performance Across 8 Clients

The results of the 8-client experiment demonstrate that increasing the network size from 4 to 8 clients does not result in a significant loss of detection performance. Accuracy and Precision are high, and have very narrow confidence intervals, which indicates that the federated aggregation process is stable even with an increase in the number of clients. Recall rates remain lower than Precision rates as they did in the smaller-scale experiment and are not reduced further with the increased number of clients.

Overall, the 8-client experiment shows that the proposed FCDA framework scales well beyond the initial 4 clients configuration, and provides a reliable and consistent detection performance.

4.7.3 Experiment 2: 16-Client Deployment

The second scalability experiment further tested the proposed FCDA architecture in an expanded federated testbed comprising 16 client nodes. The network topology comprised a single ryu controller with four OpenFlow switches linked together in a linear configuration. 4 clients were attached to each switch, yielding a total of 16 distributed clients.

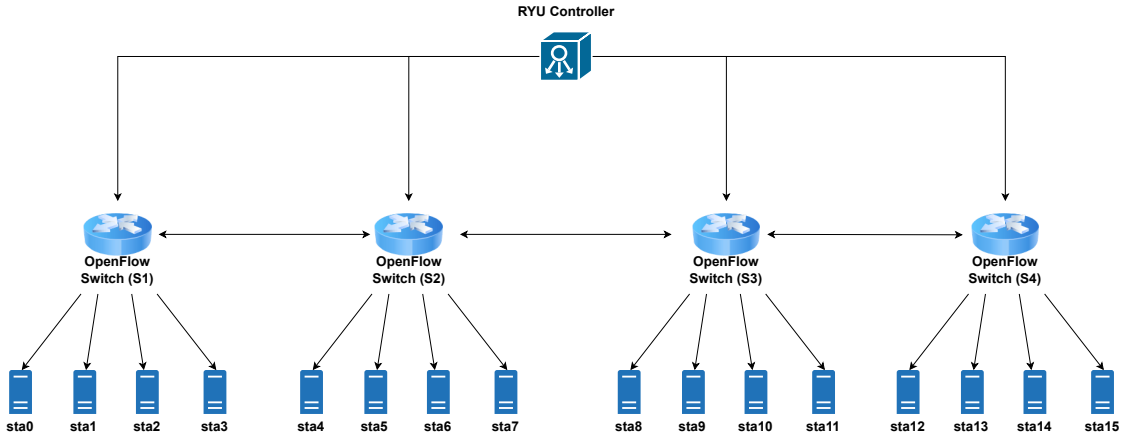

Figure 4.17 Sixteen Client Topology

The experimental methodology was the same as that used in the previous scalability experiment described in Section 4.7.2 except that we replicated the previously generated off-line deployment files to support an additional 12 clients (from sta4 through sta15). This experiment was executed for 300 FL rounds.

The final-round detection performance across all 16 clients is summarized in Table 4.15. The values provided represent the mean detection performance of the clients at the final evaluation round along with the corresponding 95% confidence interval.

These results show that increasing the size of the network from 8 clients to 16 clients resulted in very little loss of detection performance. Accuracy remained at a high level and exhibited extremely small confidence intervals; therefore, there was no variation in the classification decisions made by any of the clients. Precision and Recall followed the same patterns as the two previous experiments; therefore,

Metric	Mean ± 95% CI
Accuracy	0.953 ± 0.000
Precision	0.957 ± 0.005
Recall	0.879 ± 0.006
F1-Score	0.916 ± 0.001

Table 4.15 Mean Detection Performance Across 16 Clients

the Precision was always greater than the Recall due to the conservative anomaly detection strategy employed by the autoencoder-based model.

In summary, the results of the 16-client experiment demonstrate that the FCDA architecture provides stable and reliable detection performance regardless of the number of clients increases significantly. These findings suggest that the federated aggregation mechanism and offline initialization strategy provide effective generalization of the FCDA architecture to larger networks while still providing adequate performance within the limitations of the experimental resources.

4.7.4 Summary

Table 4.16 and Figure 4.18 present the detection performance of the FCDA framework as the number of clients increased from 4 to 8 and 16. The data shows that the performance of the system is very consistent even though all deployments are utilizing the same offline-generated artifacts from the original four-client setup.

Table 4.16 demonstrates a small decrease in mean detection accuracy from 0.960 in the four-client deployment down to 0.956 in the eight-client configuration and finally 0.953 in the sixteen-client deployment. These results show that there was an absolute decrease in detection accuracy of less than 1%, which indicates that the federated aggregation process continues to remain stable and accurate as the number of clients increases. Precision continues to remain very high across all three configurations ($\geq$ 0.957) and suggests that the addition of multiple clients to the federation does not produce additional false positives.

Recall was also consistent across deployments and was near 0.88 for all federation sizes. This behavior confirms that the conservative detection characteristics identified in smaller-scale experiments were maintained in large-scale configurations without further degradation. The resulting reduction in F1-scores was thus consistent and predictable and resulted from a balance between Precision and Recall rather than any instability or oscillations.

Number of Clients	Accuracy ± 95% CI	Precision ± 95% CI	Recall ± 95% CI	F1-Score ± 95% CI
4	0.960 ± 0.001	0.978 ± 0.005	0.885 ± 0.006	0.930 ± 0.001
8	0.956 ± 0.001	0.968 ± 0.002	0.880 ± 0.006	0.922 ± 0.003
16	0.953 ± 0.000	0.957 ± 0.005	0.879 ± 0.006	0.916 ± 0.001

Table 4.16 Detection Performance Across Network Sizes

Figure 4.18 provides additional evidence of this trend by showing the mean detection accuracy at the final round for each federation size. The downward trend in mean detection accuracy as the federation size increases clearly shows that the proposed

method is scalable and that the use of previously initialized offline-generated knowledge allows for effective growth of the federation without having to retrain offline for each scale-up.

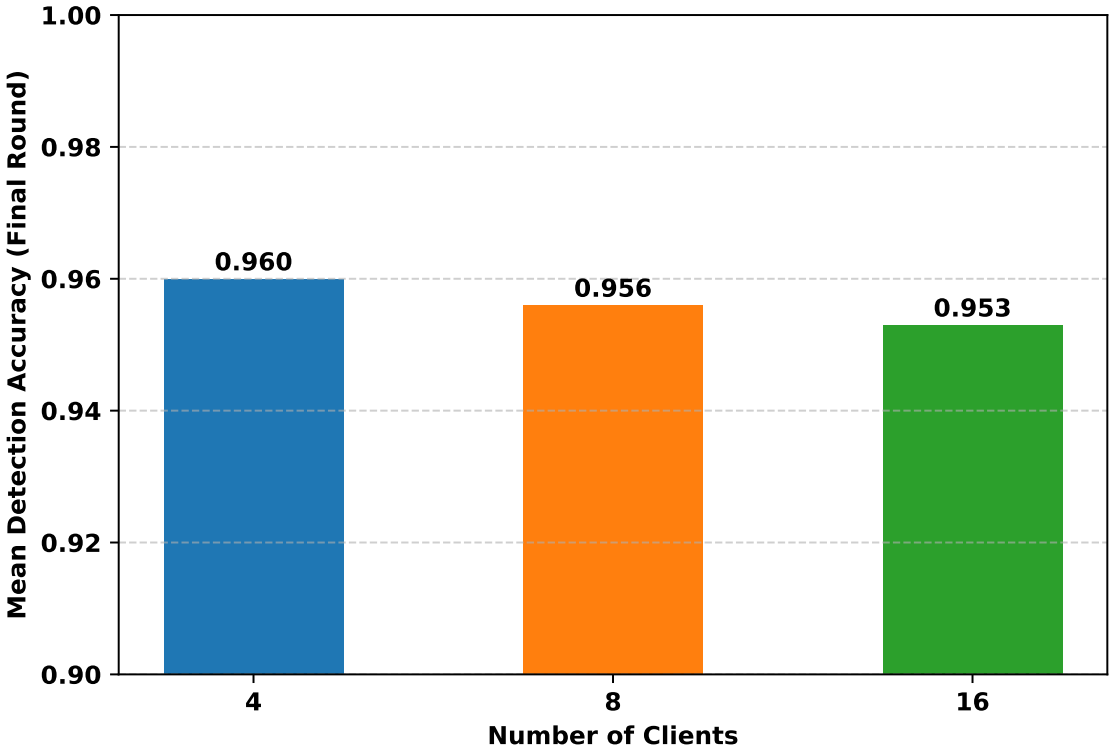

Figure 4.18 Mean Detection Accuracy vs Number of Clients

Overall, these results demonstrate that the FCDA framework can grow with the number of clients participating and that it performs well regardless of the number of clients, and that the feasibility of using the proposed design in large SDN-based IoT deployments where retraining for every scale increase would be infeasible.

5

Conclusion

This thesis addressed the challenge of detecting cyber threats in SDN-IoT networks, particularly in resource-constrained conditions. Current intrusion detection systems typically use either only network traffic analysis or only a centralized learning paradigm, neither of which adequately address the characteristics of current IoT networks: many different types of heterogeneous devices; limited computing capabilities; and distributed control. Therefore, motivated by these challenges, the focus of this thesis has been to improve SDN-based security through the combination of network traffic analysis, hardware resource monitoring, and FL.

To accomplish this objective, the thesis proposed the FCDA, a distributed anomaly detection framework intended to be used to enhance an existing CDA. The FCDA introduces a two-phase design; it includes an offline knowledge generation and online federated operation. During the offline phase, representative normal and attack data is gathered, processed, and used to train an autoencoder-based anomaly detection model and create deployment artifacts, including model weights, normalization parameters, and detection thresholds. During the online stage, these artifacts enable the instant detection of anomalies at the client level, while the FL supports continuous adaptation without centralizing raw data. Furthermore, the FCDA operates independently of mitigation actions and is designed to inform the CDA of detected attacks, thus maintaining a separation of responsibility between detection and response.

The proposed architecture was evaluated in an experimental environment developed using MininetFed, which included resource-constrained client hosts and a remote SDN controller. The test results showed that the offline-initialized FCDA achieved high and stable detection capability across multiple federated clients. Across multiple experimental runs, the FCDA consistently maintained approximately 96% average detection accuracy, high precision and a low false positive ratio, and exhibited resilient performance even under non-identical data distributions. ROC analysis provided strong evidence of the ability of the FCDA to separate normal and attack behavior, with an Area Under the Curve close to 0.98. More importantly, the

FCDA did not significantly increase the runtime CPU or memory usage of the client during online operation, thereby validating the feasibility of deploying the FCDA continuously in constrained environments.

A comparative evaluation of the FCDA with a FL configuration that started from scratch validated the importance of the offline generated knowledge. When there was no pre-trained model and no normalized values available to start the learning process, the baseline system failed to learn effectively and produced very poor detection results. This comparative validation demonstrates that offline knowledge is a critical component of achieving effective federated anomaly detection when local data availability and computational resources are limited.

While the initial four-client configuration was used to create the knowledge base for the FCDA, the effectiveness of the FCDA was also tested in larger federations of 8 and 16 clients. The scalability tests demonstrated that as the number of participating clients in the federation increases, the detection performance of the FCDA decreases gradually but minimally. Specifically, the mean detection accuracy of the FCDA decreased from approximately 0.960 with 4 clients to 0.956 with 8 clients and to 0.953 with 16 clients. Precision and recall also demonstrated similar stable trends. The results of these scalability tests demonstrate that the federated aggregation process can be reliably applied even in larger federations and that the proposed framework can scale well without the need for repeated offline training.

Despite these positive outcomes, some limitations were observed. The use of deep learning frameworks like `PyTorch` imposed a non-negligible memory footprint, making it difficult to reliably operate under extreme memory constraints (i.e., 50 MB). Consequently, all experiments were conducted with client memory limits of 512 MB to ensure the stability of the experiment. Although the online evaluation demonstrated scalability up to 16 client devices, the offline knowledge generation phase was performed utilizing a smaller set of representative client devices. Therefore, further investigation is needed to determine the performance of the FCDA in very large-scale deployments that involve tens or hundreds of devices. Furthermore, even though the FCDA can identify anomalies and generate alerts, the operational connection between the FCDA and the CDA for automated mitigation was not implemented or evaluated, and therefore remains conceptual in this work. Additionally, the evaluation was performed in a wired Ethernet-based MininetFed environment, and wireless communication scenarios have not been explored.

Another observed limitation of the FCDA relates to its conservatively defined detection behavior. As previously mentioned in the attack-type error analysis, the FCDA prioritizes precision in its detection to prevent unnecessary disruptions in resource-constrained IoT environments. While this approach will result in the FCDA being reliable and stable, it will also produce lower levels of recall for certain types of attacks, particularly UDP floods. This implies that the FCDA may not consistently identify all low intensity or short duration attacks as they do not consistently have enough anomalous characteristics to meet the threshold used by the FCDA. Most importantly, the false negatives produced by the FCDA are systematic, rather than random, and indicate a predictable trade-off in detection strategy as opposed to instability in the model.

The above limitations suggest several promising directions for future work. Techniques for compressing lightweight models or alternative inference runtimes could

lower the memory requirements of the FCDA and allow it to deploy on more resource-constrained devices. An extension of the evaluation to larger, more complex SDN topologies, including multi-controller and wireless IoT environments, will allow for better understanding of scalability and coordination effects in the FCDA. Additionally, incorporating more sophisticated and application-layer attacks (e.g. HTTP floods, Slowloris attacks, DNS amplification, etc.) into the attack model will increase the scope of the proposed framework. These attacks often produce subtle traffic patterns and resource exhaustion effects over time, thereby requiring enhanced feature representations and adaptable thresholds. Finally, implementing and evaluating real-time coordination protocols between the FCDA and CDA would enable closed-loop detection and mitigation.

In conclusion, this thesis has demonstrated that the combination of offline knowledge generation with online FL and multi-dimensional monitoring is an effective and practical method for detecting cyber threats in SDN-based IoT environments. Through the decoupling of detection and mitigation and the ability of the FCDA to learn continuously in a decentralized manner, the proposed FCDA contributes a scalable and expandable foundation for increasing the resilience of SDN.

Reproducibility.

All scripts developed and used for data processing, model training, evaluation, and plot generation in this thesis are publicly available in a GitHub repository: `https://github.com/olidsayed/FCDA`.

The repository contains the complete experimental pipeline and enables reproduction of the results presented in this work.

Bibliography

[1] Ali, S., Alvi, M. K., Faizullah, S., Khan, M. A., Alshanqiti, A., and Khan, I. Detecting ddos attack on sdn due to vulnerabilities in openflow. In *2019 International Conference on Advances in the Emerging Computing Technologies (AECT)* (2020), IEEE, pp. 1–6.

[2] Azad, K. M. S., Hossain, N., Islam, M. J., Rahman, A., and Kabir, S. Preventive determination and avoidance of ddos attack with sdn over the iot networks. In *2021 International Conference on Automation, Control and Mechatronics for Industry 4.0 (ACMI)* (2021), IEEE, pp. 1–6.

[3] Bera, S., Misra, S., and Vasilakos, A. V. Software-defined networking for internet of things: A survey. *IEEE Internet of Things Journal 4*, 6 (2017), 1994–2008.

[4] Bhayo, J., Hameed, S., and Shah, S. A. An efficient counter-based ddos attack detection framework leveraging software defined iot (sd-iot). *IEEE Access 8* (2020), 221612–221631.

[5] Biondi, P., and the Scapy Community. Scapy: The python-based interactive packet manipulation program. `https://scapy.net`, 2024. Version 2.6.1.

[6] Braga, R., Mota, E., and Passito, A. Lightweight ddos flooding attack detection using nox/openflow. In *IEEE Local Computer Network Conference* (2010), IEEE, pp. 408–415.

[7] Deng, S., Gao, X., Lu, Z., Li, Z., and Gao, X. Dos vulnerabilities and mitigation strategies in software-defined networks. *Journal of Network and Computer Applications 125* (2019), 209–219.

[8] Han, T., Jan, S. R. U., Tan, Z., Usman, M., Jan, M. A., Khan, R., and Xu, Y. A comprehensive survey of security threats and their mitigation techniques for next-generation sdn controllers. *Concurrency and Computation: Practice and Experience 32*, 16 (2020), e5300.

[9] Hsu, T.-M. H., Qi, H., and Brown, M. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335* (2019).

[10] Kalkan, K., Gür, G., and Alagöz, F. Sdnscore: A statistical defense mechanism against ddos attacks in sdn environment. In *2017 IEEE symposium on computers and communications (ISCC)* (2017), IEEE, pp. 669–675.

[11] Kandoi, R., and Antikainen, M. Denial-of-service attacks in openflow sdn networks. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)* (2015), IEEE, pp. 1322–1326.

[12] Kloth, S., **RETTORE, P. H. L.**, Zissner, P., Santos, B. P., and Sevenich, P. Towards a cyber defense system in software-defined tactical networks. In *2024 International Conference on Military Communication and Information Systems (ICMCIS)* (2024), pp. 1–8.

[13] Kloth, S., **Rettore, Paulo H. L.**, Zissner, P., dos Santos, B. P., and Sevenich, P. Securing Software-Defined tactical networks: A cyber defense system. In *2025 IFIP Networking Conference (IFIP Networking) (IFIP Networking 2025)* (Limassol, Cyprus, 5 2025), pp. 1–8.

[14] Li, C., Qin, Z., Novak, E., and Li, Q. Securing sdn infrastructure of iot–fog networks from mitm attacks. *IEEE Internet of Things Journal 4*, 5 (2017), 1156–1164.

[15] Luo, H., Li, W., Qian, Y., and Dou, L. Mitigating sdn flow table overflow. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)* (2018), vol. 1, IEEE, pp. 821–822.

[16] Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., and Elovici, Y. N-baiot—network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing 17*, 3 (2018), 12–22.

[17] Mousavi, S. M., and St-Hilaire, M. Early detection of ddos attacks against sdn controllers. In *2015 international conference on computing, networking and communications (ICNC)* (2015), IEEE, pp. 77–81.

[18] Olanrewaju-George, B., and Pranggono, B. Federated learning-based intrusion detection system for the internet of things using unsupervised and supervised deep learning models. *Cyber Security and Applications 3* (2025), 100068.

[19] Qian, Y., You, W., and Qian, K. Openflow flow table overflow attacks and countermeasures. In *2016 European Conference on Networks and Communications (EuCNC)* (2016), IEEE, pp. 205–209.

[20] Sanfilippo, S. hping3: Tcp/ip packet assembler/analyzer. `http://www.hping.org`, 2004. Version 3.0.0-alpha-2.

[21] Sarmento, E. M., Bastos, J. J., Villaca, R. S., Comarela, G., and Mota, V. F. Mininetfed: A tool for assessing client selection, aggregation, and security in federated learning. In *2024 IEEE 10th World Forum on Internet of Things (WF-IoT)* (2024), IEEE, pp. 1–6.

[22] Wang, S., Gomez, K., Sithamparanathan, K., Asghar, M. R., Russello, G., and Zanna, P. Mitigating ddos attacks in sdn-based iot networks leveraging secure control and data plane algorithm. *Applied Sciences 11*, 3 (2021), 929.

[23] Xu, T., Gao, D., Dong, P., Zhang, H., Foh, C. H., and Chao, H.-C. Defending against new-flow attack in sdn-based internet of things. *IEEE Access 5* (2017), 3431–3443.

[24] Xu, Y., and Liu, Y. Ddos attack detection under sdn context. In *IEEE INFOCOM 2016-the 35th annual IEEE international conference on computer communications* (2016), IEEE, pp. 1–9.

[25] Zhou, Y., Chen, K., Zhang, J., Leng, J., and Tang, Y. Exploiting the vulnerability of flow table overflow in software-defined network: Attack model, evaluation, and defense. *Security and Communication Networks 2018* (2018), 1–15.

List of Figures

List of Tables