TU WIEN Informatics

Semi-supervised federated learning based intrusion detection in a heterogeneous industrial IoT setting

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Dominik Rieser, BSc
Matrikelnummer 11721035

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ. Prof. Dr. Schahram Dustdar
Mitwirkung: Dr. Andrea Morichetta

Wien, 29. April 2025

Dominik Rieser	Schahram Dustdar

TU WIEN Informatics

Semi-supervised federated learning based intrusion detection in a heterogeneous industrial IoT setting

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Dominik Rieser, BSc
Registration Number 11721035

to the Faculty of Informatics

at the TU Wien

Advisor: Univ. Prof. Dr. Schahram Dustdar
Assistance: Dr. Andrea Morichetta

Vienna, April 29, 2025

Dominik Rieser	Schahram Dustdar

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Declaration of Authorship

Dominik Rieser, BSc

I hereby declare that I have written this Diploma Thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work - including tables, maps and figures - which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.
I further declare that I have used generative AI tools only as an aid, and that my own intellectual and creative efforts predominate in this work. In the appendix "Overview of Generative AI Tools Used" I have listed all generative AI tools that were used in the creation of this work, and indicated where in the work they were used. If whole passages of text were used without substantial changes, I have indicated the input (prompts) I formulated and the IT application used with its product name and version number/date.

Vienna, April 29, 2025

Dominik Rieser

Danksagung

Zunächst möchte ich meinem Betreuer Schahram Dustdar sowie meinem Co-Betreuer Andrea Morichetta für ihre Unterstützung während dieser Arbeit danken. Die Diskussions- und Feedbackrunden mit Andrea haben mich stets auf dem richtigen Weg gehalten und mir wertvolle Lektionen vermittelt.

Darüber hinaus möchte ich meinen tiefsten Dank an meine Eltern, Hermann und Sabine, aussprechen, die mich auf meinem Weg immer unterstützt und ermutigt haben. Ich bin euch für alles, was ihr für mich getan habt, ewig dankbar.

Abschließend möchte ich meinen Brüdern David und Lukas danken, die mir im Laufe meines Lebens Motivation, Inspiration und Unterstützung gegeben haben und immer für mich da waren. Ein besonderer Dank auch für euer wertvolles Feedback zu dieser Arbeit.

Acknowledgements

First, I would like to thank my advisor Schahram Dustdar and co-advisor, Andrea Morichetta, for their support throughout this work. The discussion and feedback sessions with Andrea always kept me on the right path and provided me with many valuable lessons.

Furthermore, I want to express my deepest gratitude to my parents, Hermann and Sabine, who always supported and encouraged me along the way. I am forever grateful for everything you have done for me.

Lastly, to my brothers David and Lukas, who have always provided me with motivation, inspiration, and encouragement throughout my life. A special thank you also for providing valuable feedback on this thesis.

Kurzfassung

Der Aufstieg von Industrie 4.0 hat die Entwicklung hochvernetzter intelligenter Systeme vorangetrieben, wobei Machine Learning (ML) zunehmend in anomaliebasierten Intrusion Detection Systemen (IDS) in Industrial-Internet-of-Things (IIoT)-Umgebungen eingesetzt wird. In diesem Kontext hat sich Federated Learning (FL) als dezentraler Ansatz etabliert, der Sicherheit und Datenschutz verbessert, indem ML-Modelle auf verteilten Geräten trainiert und die Ergebnisse auf einem zentralen Server aggregiert werden, ohne dabei lokale Daten zu übertragen. Traditionelle Ansätze wie Federated Averaging (FedAvg) stoßen jedoch in heterogenen Umgebungen auf Herausforderungen, insbesondere hinsichtlich der ineffizienten Ressourcennutzung und einer erhöhten Anfälligkeit gegenüber Denial-of-Service (DoS)-Angriffen aufgrund der zugrunde liegenden synchronen Aggregation. Asynchronous Federated Learning (AFL) bietet eine vielversprechende Alternative, indem es Probleme in Zusammenhang mit Geräteunzuverlässigkeit, Aggregationseffizienz und Konvergenzgeschwindigkeit mindert. Trotz dieser Entwicklungen adressieren bestehende Federated Learning-basierte IDS Frameworks die Heterogenität der Geräte in IIoT-Umgebungen weiterhin unzureichend.

Diese Arbeit schlägt ein verbessertes Federated Learning Framework für heterogene Umgebungen vor, mit besonderem Fokus auf die Erkennung von DoS-Angriffen. Zur Unterstützung des Übergangs zu einer heterogenen Systemarchitektur wird eine strukturierte Methodik erstellt, die als Grundlage für die Entwicklung des Frameworks dient. Diese Methodik wird auf ein bestehendes Federated Learning-basiertes Intrusion Detection Framework angewandt, und ein umfassendes Architekturdesign wird präsentiert, das eine einheitliche Systembereitstellung auf heterogenen Edge-Geräten, asynchrone Modellaggregation, dynamischen Datenumgang, Backup-Mechanismen sowie eine zentrale Modellevaluierung umfasst. Die vorgeschlagene Lösung wird in einem kleinen, heterogenen IIoT-Testumfeld bezüglich System- und Modellleistung empirisch evaluiert. Die Ergebnisse zeigen, dass unser Framework Leerlaufzeiten signifikant reduziert, die Update-Rate erhöht und zusätzliche systemseitige Widerstandsfähigkeit gegenüber DoS-Angriffen bietet sowie die Modellleistung und die Konvergenzrate verbessert.

Abstract

The rise of Industry 4.0 has driven the development of highly interconnected intelligent systems, with Machine Learning (ML) increasingly utilized in anomaly-based intrusion detection systems (IDS) in Industrial Internet of Things (IIoT) settings. In this context, Federated Learning (FL) has emerged as a decentralized approach that enhances security and privacy by training ML models on distributed clients and aggregating the results on a central server without sharing the underlying local data. However, traditional federated learning methods, such as Federated Averaging (FedAvg), face challenges in heterogeneous environments, including inefficient resource utilization and vulnerability to Denial of Service (DoS) attacks caused by the underlying synchronous aggregation. Asynchronous Federated Learning (AFL) presents a promising alternative, mitigating issues related to device unreliability, aggregation efficiency, and convergence speed. Despite these advancements, existing federated-learning-based anomaly detection frameworks still inadequately address device heterogeneity in IIoT settings.

This thesis proposes an improved federated learning framework for heterogeneous environments, with a particular focus on DoS attack detection. To guide the transition to a heterogeneous system architecture, we develop a structured methodology that serves as the foundation of our framework development. We apply this methodology to an existing federated-learning-based intrusion detection framework and present a comprehensive architectural design that features uniform system deployment on heterogeneous edge devices, asynchronous model aggregation, dynamic data utilization, backup mechanisms, and a centralized model evaluation. The proposed solution is empirically evaluated in a small-scale, heterogeneous IIoT testbed with respect to system and model performance. Results demonstrate that our framework significantly reduces idle times, increases update rates, and provides additional system-level resilience against DoS attacks, while also improving model performance and convergence rate.

Contents

CHAPTER 1

Introduction

1.1 Problem Statement and Motivation

Through the revolution of Industry 4.0, where devices are interconnected more than ever, intelligent systems have emerged that are capable of advanced analytics and decision-making [BMD23]. Machine learning is frequently used in modern anomaly-based Intrusion Detection Systems (IDSs) to detect ongoing attacks and enable timely responses. Federated learning [MMR+17] has emerged as an advancement in distributed machine learning, promising security and privacy-preserving characteristics. Especially in an Industrial Internet of Things (IIoT) setting, where security and privacy are central concerns, federated learning can be used to reduce the overhead of sending massive amounts of heterogeneous data to a central server [BMD23]. Federated learning allows distributed clients to train a machine learning model on a local dataset, share the resulting model parameters with a central server, which aggregates and builds a global model, and finally use the updated model for future training. The use of federated learning not only minimizes the risk of exposing confidential and sensitive information over the Internet but also reduces bandwidth costs and latency issues [BMD23].

However, traditional federated learning approaches, such as Federated Averaging (FedAvg) [LHY+19], face challenges in heterogeneous environments, as they typically update the remote model in a synchronous manner once all federated devices complete their training [XQXG23]. This leads to resource utilization limitations due to the need to wait for slow devices, also known as stragglers, before aggregation in each training round [XQXG23].

Furthermore, FedAvg has been shown to have limitations with respect to Denial of Service (DoS) attacks [DCS24]. This includes the convergence rate, as FedAvg uses an inefficient client selection process in emergency situations [DCS24]. In the context of Intrusion Detection System (IDS), it is crucial to employ systems that are able to effectively handle DoS attacks, especially zero-day attacks. A short convergence time is

essential for quickly distributing an updated global model that includes the newly gained DoS attack information [Rou21].

Asynchronous federated learning emerges as a promising solution to some of the challenges regarding heterogeneous environments, in particular the unreliability of devices, aggregation efficiency and resource utilization [XQXG23]. The asynchronous aggregation approach allows the server to aggregate the model as soon as individual clients share local updates and to initiate a new training round on the respective client. As a result, the need to wait for stragglers is eliminated, reducing idle times and increasing the update rate [XQXG23].

Current approaches in federated-learning-based anomaly detection in an IIoT setting fail to address device heterogeneity, as shown in the works of [BMD23] and [APMS22]. In response, this thesis proposes an improved framework using promising solutions presented in recent research, such as asynchronous federated learning, to optimize the system architecture for heterogeneous environments, while emphasizing on DoS attack detection. For this purpose, we utilize a small-scale heterogeneous IIoT testbed in which the federated framework is deployed and evaluated. To guide our process, we define a methodology for the transition from a homogeneous to a heterogeneous environment.

1.2 Research Questions

The aim of this thesis is to provide scientifically justified answers to following research questions:

- **RQ1: What is a representative methodology to transition a homogeneous federated intrusion detection system architecture to a heterogeneous environment?**

  This question is answered by providing a structured process of reviewing an existing system architecture regarding selected patterns and design decisions, applying these pattern while identifying potential limitations, drawbacks and trade-offs, as well as effectively evaluating the introduced architecture.

- **RQ2: What are the design implications of applying this methodology to an existing synchronous federated learning framework when focusing on effectively handling DoS attacks?**

  To answer this question, we apply the proposed methodology on a preliminary developed framework, selecting patterns and design decisions which enable us to mitigate DoS attacks while showcasing limitations and balancing trade-offs. We publish the implementation as an open-source framework, facilitating future research on this and adjacent topics.

- **RQ3: How do synchronous and asynchronous federated-learning-based intrusion detection systems compare with respect to model and system performance?**

  We answer this question by empirically evaluating the baseline (synchronous) and proposed (asynchronous) framework in a small-scale, heterogeneous IIoT testbed, showcasing the performance differences in heterogeneous environments and validating the effectiveness of our design approach.

1.3 Methodology

The goal of this thesis is to propose a framework for federated learning in a heterogeneous environment, with a focus on DoS attack detection. We select a novel framework proposed by Bekbulatova et al. [BMD23] as the baseline for our improvement efforts and intend to utilize a small-scale, heterogeneous IIoT testbed for evaluating our work.

We achieve this goal through the process of answering our research questions. This process can be structured into five stages:

1. **Relevance:** In this stage, we justify the relevance of the problem through a literature review. The result is a report on the topics of federated-learning-based anomaly detection, federated learning in heterogeneous environments, asynchronous federated learning approaches, and relevant key architectural design aspects and patterns, based on state-of-the-art literature.
2. **Architecture Design:** Based on the literature review, we define a methodology guiding our transition process from a homogeneous federated intrusion detection system architecture to a heterogeneous environment. Through this structured process we design the system architecture of our proposed framework. The result is a comprehensive architectural design that addresses device heterogeneity and ensures robustness against DoS attacks.
3. **Implementation:** This stage covers the adaptation of the existing system based on our architecture design. The result is a well-documented implementation, ready to be published as an open-source framework to facilitate future research.
4. **Evaluation:** In this stage, the developed framework is empirically evaluated against the baseline with respect to the system and model performance using key metrics. The evaluation provides a validated comparison demonstrating the improvements of our framework over the baseline.

The expected results of this thesis, along with the answers to the research questions, should enable future researchers to adapt traditional synchronous federated-learning-based intrusion detection systems for heterogeneous device environments. The proposed framework specifically addresses limitations related to device heterogeneity and DoS attack mitigation in existing approaches.

1.4 Focus

The focus of this work is on the system architecture of federated-learning-based intrusion detection systems in heterogeneous environments. Although the underlying machine learning model is the foundation for our proposed framework, model optimization itself is beyond the scope of this thesis. While heterogeneous environments generally encompass both data and device heterogeneity, our improvement efforts specifically target device heterogeneity. Addressing data heterogeneity in federated learning constitutes a complex research area that would significantly broaden the scope of this work and is therefore not considered. Additionally, this thesis concentrates on Denial of Service (DoS) attack mitigation, with other attack types only briefly discussed. Extending the framework to support multi-anomaly classification, including sophisticated attacks, represents an important next step, but it is outside the scope of this work and is suggested for future research.

1.5 Thesis Structure

This thesis is structured as follows. Chapter 2 provides an overview of the underlying concepts, including DoS attacks, IIoT, IDS, and federated learning. In Chapter 3, results of the literature review on federated-learning-based anomaly detection, federated learning in heterogeneous environments, asynchronous federated learning approaches, and relevant key architectural design aspects and patterns are presented. Chapter 4 first defines a methodology for the transition process from homogeneous to heterogeneous system architectures in federated learning. Based on this methodology, the existing framework is reviewed, and the architecture of our improved framework is designed, addressing device heterogeneity and DoS attacks. Chapter 5 presents details regarding the implementation of our framework based on the previously introduced architecture design. In Chapter 6, the proposed framework is empirically evaluated against the baseline and the results are presented. In Chapter 7, limitations of the proposed framework are examined and discussed, and Chapter 8 summarizes the main contributions and findings of this work. The thesis concludes with Chapter 9 by identifying potential avenues for future research.

CHAPTER 2

Background

2.1 Denial of Service Attacks

Most commonly, Denial of Service (DoS) is used to impair a computer or network resource from legitimate use [LT01]. Early attack schemes of DoS attacks involved sending packets from a single source computer to a single destination computer and over time, this has evolved to more sophisticated attacks involving multiple distributed sources against single or multiple targets, classified as Distributed Denial of Service (DDoS) attacks [LT01].

Common approaches for DoS attacks are sending streams of packets to a victim to consume key resources in order to make the service unavailable for legitimate use or sending malformed packets to confuse an application to force it to freeze or reboot [MR04]. According to Mirkovic et al. [MR04], attackers generally recruit agent machines through automated exploitation of security holes and infecting vulnerable machines with the attack code. The compromised agents are then often used to automatically recruit new agents and send attack packets to the victim. Through spoofing of the unregulated source address field, attackers are able to hide the identity of compromised machines. The goals of DoS or DDoS attacks can be manyfold, ranging from personal reasons, prestige to material gain or political motives [MR04].

Especially in times of conflict or war, DDoS attacks have become more prominent in order to damage and impair the opponent [TÇES25]. For example, in the recent Ukraine-Russia conflict, there have been reported DDoS attacks from both sides attacking many different sectors such as government, news, finance, business and travel [TÇES25].

2.2 Industrial Internet of Things

In [SHH+20], Internet of Things is described as the interconnection between vast amount of devices. These devices range from simple sensors to complex controllers and home

appliances, enabling many new applications, including services to enhance and automate daily tasks. For Serror et al. [SHH+20], the crucial part about these devices is that they are internet-connected, thus exchanging data with other devices and services. They further state that the reason for this emergence is the decrease of costs for hardware and software, thus making it available to private homes and allowing for a multitude of distributed devices working together. Over time, the domain of IoT spread to the industrial sector, where connecting previously isolated devices led to the fourth industrial revolution, Industry 4.0, also called IIoT [SHH+20].

However, according to Serror et al. [SHH+20], it has been shown that for its users, IoT bears substantial security and privacy risks. They explain this with either completely disregarding or poorly implementing security features, as well as not rolling out security updates and having vulnerabilities unpatched. Since the broad audience of IoT applications consists of users without a deeper knowledge of networks and internet security, these users often do not know how to configure their network properly to decrease security risks [SHH+20]. IoT devices are thus often targeted in botnet attacks such as Bashlite or Mirai [MAF+18, SHH+20].

Research in the security domain of Industry 4.0 also shows that IIoT devices are just as likely as IoT devices to be prone to security risks and vulnerabilities [WHA+16, SWW15, SSH+18]. Furthermore, attacks on industrial devices are oftentimes far more disastrous than attacks on private IoT devices, as can be seen in the attacks on the German steel mill in 2014 [LAC14] or the Ukrainian power grid in 2015 [WOGS17]. A key difference for IoT and IIoT devices according to Serror et al. [SHH+20] is that IIoT devices have a much longer lifetime, namely up to 10-30 years in contrast to consumer devices with an average lifetime of 3-5 years. Thus, they infer that IIoT devices are required to adapt to changing environments regarding security management and regularly patching vulnerabilities. In addition, IIoT environments are typically larger than consumer environments, making it more difficult to maintain a secure network architecture [SHH+20].

IoT attacks can be classified depending on a layered architecture model, as often described in recent research [MYAZ15, YWY+17]. According to Serror et al. [SHH+20], this model consists of perception, network and architecture layer, each with different scopes, common attacks and distinctive countermeasures. In the perception layer physical or impersonation attacks can occur, where attackers are either gaining physical access to the device or create a false identity through spoofing. Prominent attacks in the network layer are Man-in-the-Middle (MitM) and routing attacks. Serror et al. [SHH+20] state that in MitM attacks, an attacker can gain access to sensitive information through interception of communication between two parties. The application layer focuses on the software of the IoT devices and is especially challenging to secure because new vulnerabilities can be introduced through software changes. Common attacks in this layer are malicious code injection and data leakage. Serror et al. [SHH+20] further state there exist attacks, such as DoS, which can be found in multiple layers. According to them, the primary goal of DoS attacks on the perception layer is to interrupt the communication or sensing abilities of targeted IoT devices. On the other hand, the objective of DoS attacks on

the network layer is to disconnect devices that are responsible for the communication infrastructure, e.g. routers, while on the application layer its target is to flood critical services with requests [SHH+20].

Countermeasures for securing IIoT can be implemented with different approaches, from cryptography and authentication, patch management, service isolation, network monitoring and intrusion detection to awareness, training and assessment in the corporate culture [SHH+20]. In the following section intrusion detection will be explored in-depth.

2.3 Intrusion Detection

According to Serror et al. [SHH+20], IDSs have been a crucial security measure to detect ongoing attacks enabling timely responses. One of its key applications is to deal with zero-day vulnerabilities, i.e. a vulnerability that is unknown to the vendor and for which there is no security patch available [Rou21, SHH+20]. Serror et al. [SHH+20] state that IDSs require efficient network-based monitoring in real-time, while not interfering with crucial processes. Because of the inherent tasks in IIoT environments, regular network traffic patterns can be examined and thus creating new possibilities in anomaly and intrusion detection [SHH+20]. Serror et al. [SHH+20] examine that using a model of the industrial process can further reduce the detection error rate and categorize such systems as process-aware intrusion detection systems.

According to Khraisat et al. [KGVK19], IDSs can be classified as signature-based or anomaly-based. In signature-based detection a pattern of an attack or threat is used to compare and match it against captured events. They state that is a simple but effective method, often used for detecting already known attacks, but struggles especially with unknown attacks and keeping the patterns up to date. Anomaly-based intrusion detection systems on the other hand, which compare the normal and expected behavior profiles against observed events, are most effective for new and unforeseen vulnerabilities [KGVK19].

Instead of classifying by detection method, another way to classify IDSs is based on the input data sources [KGVK19]. According to Khraisat et al. [KGVK19], the two main types in this category are Host-based Intrusion Detection Systems (HIDS) and Network-based Intrusion Detection Systems (NIDS). HIDSs monitor data from host systems like operating systems, windows server logs, firewall logs and application system audits. They are effective for handling insider threats that do not involve network traffic. NIDSs, on the other hand, analyze network traffic through network packet capture to detect external threats early. A limitation of NIDS, according to Khraisat et al. [KGVK19], exists in high bandwidth environments, where they might not be able to monitor all data in a network due to the sheer volume of data passing through the network. One possible approach to mitigate this limitation is to deploy the NIDS at multiple positions in the network, and additionally use HIDS and firewalls in order to provide protections against both internal and external attacks [KGVK19].

2.4 Federated Learning

Federated learning is an advancement in distributed machine learning first introduced in 2017 by McMahan et al.[MMR+17], which allows collaboratively training a model on many distributed clients. In contrast to distributed machine learning, which can generally be categorized as centralized, decentralized and fully distributed, FL is designed for environments where data privacy is crucial, whereas distributed machine learning often assumes cloud or data center environments [XQXG23]. Examples for such privacy-preserving environments are mobile phones or IIoT settings. In Federated learning, each client holds its own dataset on which the model is trained and shares the resulting model parameters to a central server [BMD23]. The central server then aggregates the results and builds a global model and distributes the new model to the clients for future training. A major advantage of this approach is the inherently privacy preserving nature of federated learning due to the reduction of transmitted data to the central server and sharing only model updates [XQXG23, BMD23]. The use of federated learning reduces the vulnerabilities of transferring confidential and sensitive data over the internet and further decreases bandwidth costs and latency issues [BMD23].

According to Kairouz et al. [KMA+21], a typical federated training process is orchestrated by a server and involves a repetition of the following steps:

- **Client selection:** The server selects a subset of clients that meet eligibility requirements. These requirements could be set in a way to not interfere with normal operations, for example, if the device is in use or under heavy load. Additionally, for mobile applications, a mobile phone might only be eligible if it is connected to power.
- **Broadcast:** In this step the participating clients download the current model weights from the server.
- **Client computation:** The clients now execute the training program locally on their respective dataset and update the local model.
- **Aggregation:** All updates from participating clients are collected and aggregated by the server.
- **Model update:** The server locally computes a new model based on the aggregated results from the clients.

An expectation which often does not hold true in real world applications is that in federated learning, datasets across nodes are independent and identically distributed, also referred to as IID [XQXG23]. According to Xu et al. [XQXG23], in practice, datasets usually deviate from this expectation, which gives rise to a number of challenges across federated learning approaches. As an example, they describe a federated system across multiple hospitals. IID would assume that disease cases in one hospital show similarities

to disease cases in another hospital and could be regarded as the same data. Non-IID however assumes the diversity of such disease cases, which is often the case in real-world scenarios [XQXG23].

The first and widely adopted and prevalent algorithm in federated learning is FedAvg [LHY+19]. It runs Stochastic Gradient Descent (SGD) on a subset of clients in parallel and then averages the results on the central server. According to Xu et al. [XQXG23], as FedAvg inherently updates the remote model in a synchronous manner after all federated devices complete their training, this leads to challenges if the edge devices are not homogeneous. Because a new round only starts after a previous round has been completed on all other participating devices, slow devices, also known as stragglers, create a bottleneck on the system, leading to resource utilization limitations [XQXG23]. Xu et al. [XQXG23] further state that this challenge is not limited to heterogeneous devices, but is also apparent in heterogeneous data environments, where the training data distribution is uneven across devices. Additionally, it has been shown that the current node selection algorithm is inefficient, as it often causes not selecting competent devices for a participation round [XQXG23].

In regards to effective threat detection, especially for DDoS attacks, Doriguzzi et al. [DCS24] argue, that FedAvg does not satisfy two crucial requirements. First, a short convergence time is needed to reach the desired accuracy and to quickly distribute the global model to clients with newly gained DDoS attack information. FedAvg does not differentiate how each client's data influences the global model's accuracy and assigns the same amount of computation to all clients. According to Doriguzzi et al. [DCS24], this can lead to training rounds that do not substantially increase the accuracy of the system. Secondly, since FedAvg uses weighted averaging, it favors clients with large datasets and thus can not successfully adapt to detect attacks only seen in small training sets, assuming a non-IID environment [DCS24].

2.4.1 Asynchronous Federated Learning

Asynchronous Federated Learning (AFL) emerges as a promising solution to some of the challenges mentioned in 2.4, in particular unreliability of devices, aggregation efficiency and resource utilization [XQXG23]. In contrast to FL, AFL does not wait for a full round of training to be completed, but rather aggregates the model upon each reception of local updates, thus improving aggregation efficiency, and then either instantly initiates a new training round on this particular device or schedules a task for the future [XQXG23]. Resource utilization is also improved due to the reduction of idle times on clients in heterogeneous environments [XQXG23]. This approach and the comparison between traditional Federated Learning is visualized in Figure 2.1, showing that the global model is updated much more frequently and there is no waiting period for faster devices finishing their training round earlier than slower ones.

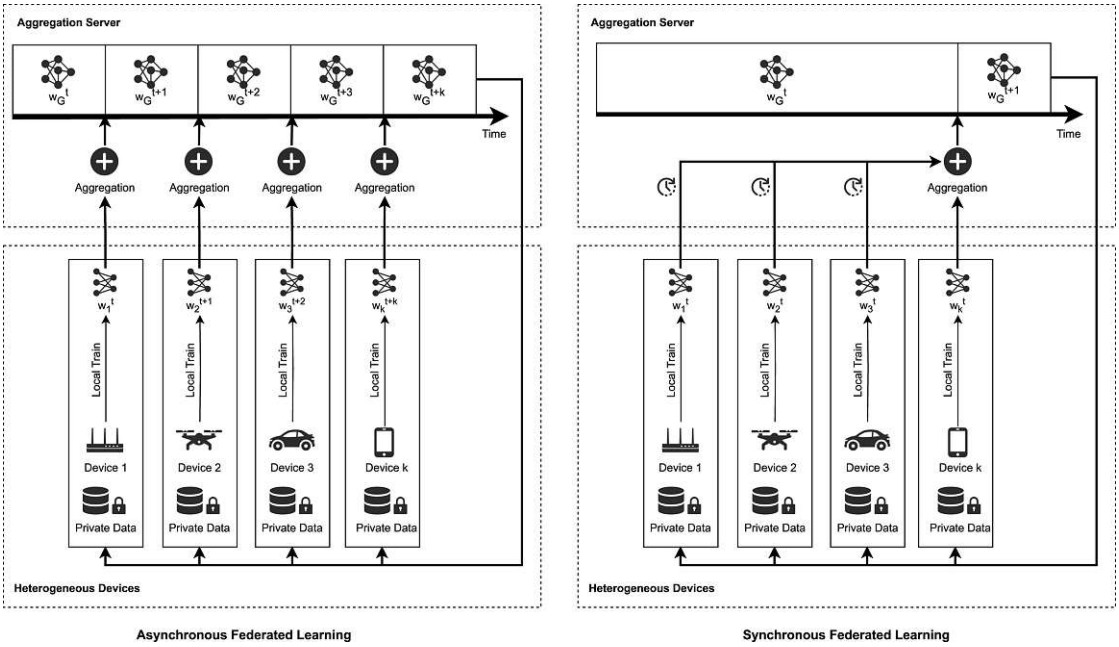

Figure 2.1: Comparison between Asynchronous Federated Learning and Synchronous Federated Learning workflows on heterogeneous devices [XQXG23].

2.4.2 Architectural Design Aspects and Patterns

According to Xu et al. [XQXG23], due to the non-IID nature of federated learning, a distinction can be made between Horizontal Federated Learning (HFL) and Vertical Federated Learning (VFL), based on how data is allocated across devices with respect to the sample space and feature space, as visualized in Figure 2.2. They state that in horizontal FL, the feature space of the clients is the same, whereas the sample space differs. In vertical FL, clients have the same sample space but a different feature space. Horizontal data partitioning allows federated learning systems to deploy the same model architecture on each device to train their local model [STT23]. On the other hand, with vertical partitioning it is possible to aggregate distinct features and provide updated gradients without accessing sensitive information [XQXG23].

According to Zhang et al. [ZBO20], the most common architecture for federated learning systems is a centralized, client-server approach. In this setting there exists a single server orchestrating all clients. The server initiates training on participating devices, aggregates the resulting model updates and deploys the global model [ZBO20]. Zhang et al. [ZBO20] argue, that this approach exhibits a smooth and elegant model transmission and is easily customizable, thus making it especially interesting for small systems as it is fairly easy to set up. They state that it does, however, show scalability issues for higher number of clients and is a single point of failure. An example of a centralized aggregation architecture is FedAvg, and many other proposed frameworks in recent years build upon this approach [LHZ+22]. Figure 2.3 shows a visualization of this and the

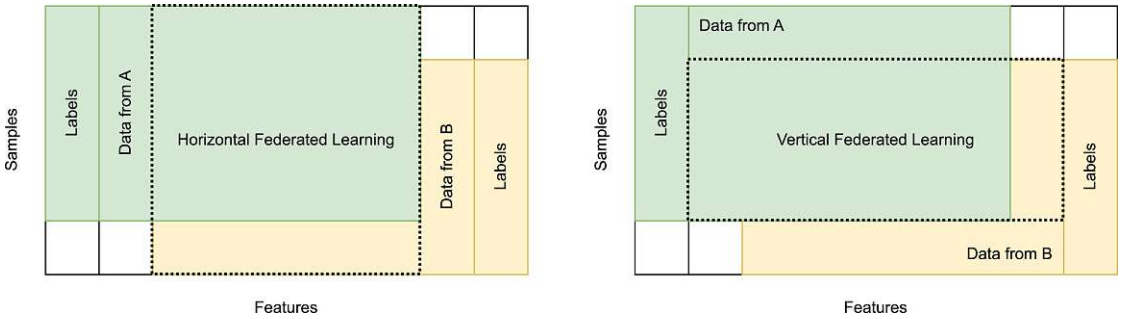

Figure 2.2: Comparison between Horizontal Federated Learning (HFL) and Vertical Federated Learning (VFL), visualized by [XQXG23].

following architectures, created by Zhang et al. [ZBO20].

According to Liu et al. [LHZ+22], a step towards decentralization can be found in hierarchical aggregation architectures, in which multiple servers are employed to reduce the model transfer time between server and clients. They state that this architecture still builds upon a global server, but uses regional server to aggregate results from its cluster. Regional servers aggregate results and send them to the global server, which returns a global model of all aggregated regional results to distribute to the clients. This approach is used for applications with large regional differences between devices, for example, in a mobile environment. In this case, the regional servers are deployed at cell base stations, where mobile phones can be connected [LHZ+22].

According to Zhang et al. [ZBO20], there also exists an architecture which can be categorized between hierarchical and fully decentralized, the regional architecture. It is similar to hierarchical in that it uses regional servers to aggregate results from its edge cluster, but it does not involve a global aggregation server. The main advantages of this approach are to eliminate the central server, as it is a single point of failure, and increase system robustness, as well as computational efficiency [ZBO20].

Zhang et al. [ZBO20] further state that a fully decentralized architecture does not rely on any aggregation servers and aggregates the results at the edge nodes. According to them, with this approach, the number of performance bottlenecks from global servers is minimized and the autonomy and adaptability is maximized at the cost of coordination difficulties. As each node operates independently and aggregates the results from neighboring nodes, it is challenging to achieve global knowledge and collective tasks [ZBO20].

Patterns can be employed at multiple stages of the federated learning life cycle on both the server and the client device, from client management, model management, model training to model aggregation [LLZ+22]. The work by Lo et al. [LLZ+22] provides an overview of 14 patterns found in recent research and real-world applications. In the following sections, some of those most relevant to our use case will be described in more detail.

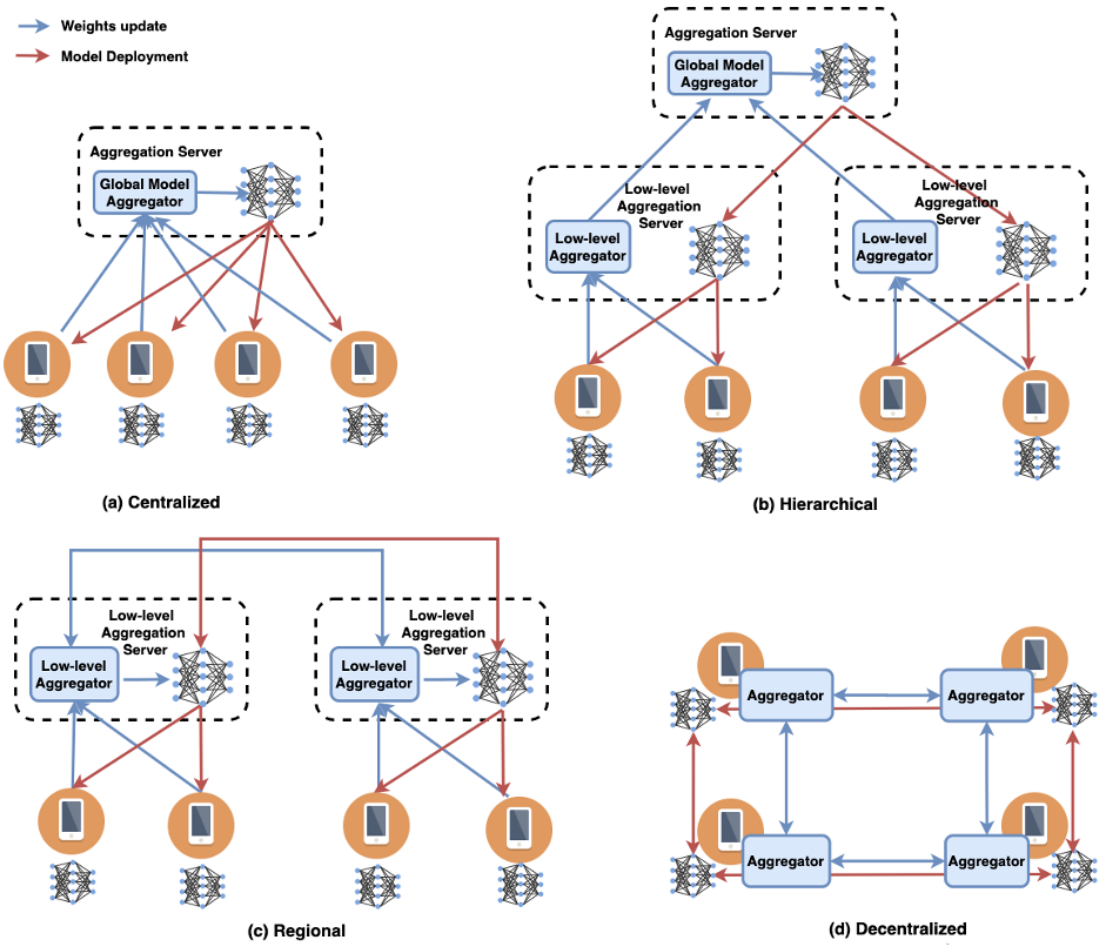

Figure 2.3: Different types of architectures from centralized to decentralized. Visualized by [ZBO20]

A **client registry**, employed at the global server, can be used to maintain information on all devices in the federated environment [LLZ+22]. According to Lo et al. [LLZ+22], this pattern is useful for distributing training tasks and keeping track of available and active nodes. Especially in federated settings, where continuous connectivity and availability are not guaranteed, clients can fail or drop out at any moment. In a synchronous learning setting this is crucial, as the server needs to know when a round is completed. A client registry thus allows for a more maintainable and reliable system, because it allows the server to manage connected and disconnected nodes, as well as providing information on problematic nodes [LLZ+22]. However, a drawback of this approach is that client information held at the sever may cause data privacy issues, which could potentially lead to the deduction of individual user usage patterns [LLZ+22].

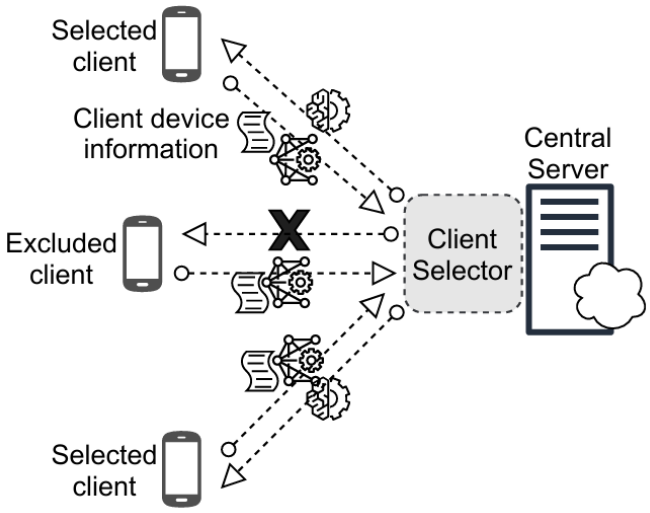

Figure 2.4: A visualization of the client selector pattern by Lo et al. [LLZ+22].

A pattern that is also based on the evaluation of information from individual clients is the **client selector**, as seen in Figure 2.4 [LLZ+22]. According to Lo et al. [LLZ+22], it can use certain information, such as available resources, data quality, and model performance, from individual clients to determine which clients should be selected for the next training task. They further state that in an environment with massive amounts of clients, the server can select only the best fitting clients, which reduces communication costs with low-quality devices and additionally increases model and system performance. Similarly to a client registry, it is prone to data privacy issues and the loss of potentially essential data [LLZ+22]. However, they state that in a setting where each client device's local data is equally essential, it is difficult to use a client selector effectively without losing data from weaker clients.

According to Lo et al. [LLZ+22], in the context of model management, a pattern to reduce communication cost is the **message compressor**. As the system scales up in terms of client devices, more communication for model updates occurs between the server and the clients. The authors further state that in order to avoid this becoming a critical bottleneck, messages between client and server can be compressed before transmission. For example, Google proposes two update approaches called structured update and sketched update to increase communication efficiency [LLZ+22].

Due to inherently non-IID environments in real life, a **heterogeneous data handler** can solve the issues of skewed data through data volume and data class addition, while maintaining local data privacy [LLZ+22]. Lo et al. [LLZ+22] state that the handler generates augmented data on the client until the volume is balanced across all devices and the classes in the datasets are equal. This is achieved by periodically gaining

information from other devices without accessing their data directly. Although it increases the performance and generality of the model, the computational cost of dealing with heterogeneous data is substantial and should not be disregarded [LLZ+22].

According to Lo et al. [LLZ+22], an **asynchronous aggregator** specifically tackles the challenges of heterogeneous device environments. They further state that as each device has different computational resources and thus different model training times, conventional federated learning settings with a synchronous model aggregator show increased waiting times until all updates are returned and a model aggregation can be performed. With the use of an asynchronous aggregator, global model updates occur as soon as a model update is received from a client and consequently, a new round of training is initiated instantly. However, a drawback of this approach is the model bias, as each global model update does not include all local model updates and additionally, computationally better clients send more updates and thus might be favored [LLZ+22].

In addition to proprietary federated learning frameworks, such as NVIDIA Clara Train, there exist several open-source federated learning frameworks, such as Flower[1], TensorFlow Federated[2], FATE[3], and PySyft[4]. Each framework has its peculiarities and offers specific features. According to [KYF+20], differentiating characteristics can be observed in supported operating systems, documentation, settings, data types and partitioning, modes, and protocols. The decision should therefore be made in the context of the use case and the objectives of the application.

[1] https://flower.ai/
[2] https://www.tensorflow.org/federated
[3] https://github.com/FederatedAI/FATE
[4] https://github.com/OpenMined/PySyft

CHAPTER 3

Related Work

The work by Bekbulatova et al. [BMD23] is closely related to this work, as it is used as the baseline for our work with the goal in mind to improve upon the proposed approach regarding the heterogeneity of edge devices in a real-life testbed and mitigating DoS attacks in the system. The recently published paper presents a framework for semi-supervised anomaly detection in an Industrial IoT setting. It shows that the use of federated instead of centralized learning is a promising approach with highly valuable security and privacy preserving characteristics, while having minimal performance differences. However, the benchmarks are run in a virtual homogeneous environment on a single machine disregarding any kind of device heterogeneity. Furthermore, it only focuses on a single kind of attack, Denial of Service (DoS), although the underlying dataset also contains Command Injection, Reconnaissance and Backdoor attacks. It was developed using the federated learning framework Flower [BTM+20] and uses Deep-SAD [RVG+19] as its deep learning model. As anomaly detection is usually treated as an unsupervised learning problem, the authors of Deep-SAD argue that in practice there often exists a small set of labeled samples and thus they provide this model for semi-supervised anomaly detection.

The underlying dataset used by Bekbulatova et al.[BMD23], as well as in this work, is WUSTL-IIOT-2021 [M. 21]. It simulates an industrial water supervision system consisting of data from a water tank, sensors, actuators, and the underlying Modbus network, in which different cyber-attacks are performed in the span of 52 hours [BMD23]. Table 3.1 shows general information about the dataset. It contains 1,194,464 data samples, in which 87,016 samples represent attacks and 1,107,448 represent normal traffic. Table 3.2 shows the distribution of traffic samples. The total attack traffic is 7.28%, of which 89.98% represent DoS traffic. This was deliberately designed imbalanced, to closer represent real-life environments, as DoS attacks are usually traffic intensive, while attacks such as command injection or backdoor only occur in a small number of traffic data [M. 21]. The 41 selected features were defined as being common in network flows and changing during

Dataset	WUSTL-IIoT
Number of observations	1,194,464
Number of features	41
Number of attack samples	87,016
Number of normal samples	1,107,448

Table 3.1: Specifics of the WUSTL-IIOT Dataset. [M. 21]

Traffic type	Percentage (%)
Normal Traffic	92.72
Total Attack Traffic	7.28
Command Injection Traffic	0.31
DoS Traffic	89.98
Reconnaissance Traffic	9.46
Backdoor Traffic	0.25

Table 3.2: Traffic types and respective percentages in the WUSTL-IIOT Dataset. [M. 21]

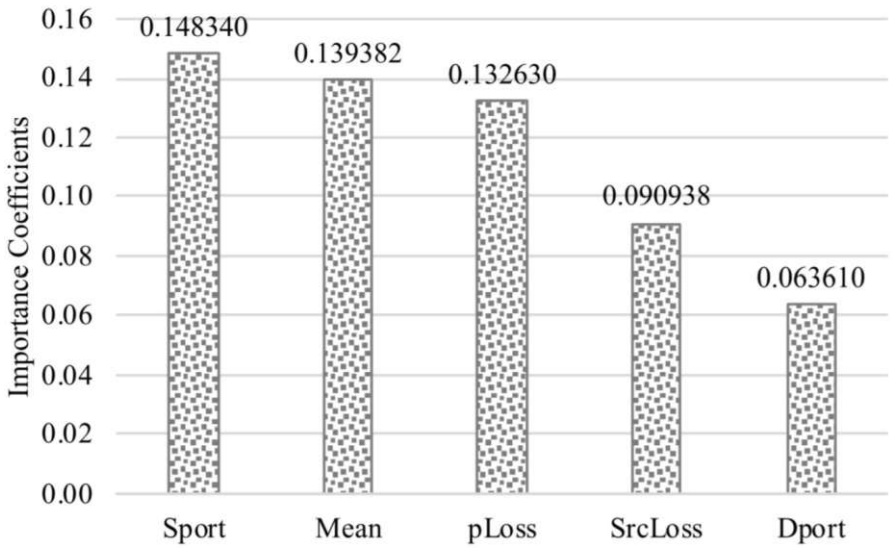

Figure 3.1: Normalized importance of the five most impactful features in WUSTL-IIoT Dataset [M. 21].

attacks [M. 21]. According to the authors of the dataset [M. 21], although all features show importance during attacks and are relevant for training, the five most impactful features are Source Port (Sport), Mean flow (mean), Total Percent Loss (pLoss), Source Loss (SrcLoss) and Destination Port (Dport), as seen in Figure 3.1. The importance coefficient represents the normalized importance of the 41 features.

In this thesis [Gaj24], Gajanin studied the transition from centralized learning to federated learning, specifically in a non-IID environment. It provides a structured and formal methodology highlighting challenges in this process, as well as providing some countermeasures. The author conducts a case study in the area of human activity recognition regarding classes such as walking, sitting or bicycling. It shows that device heterogeneity and variations in local label and feature distributions decrease the performance and stability in model training [Gaj24]. Gajanin [Gaj24] further shows that especially for asynchronous federated learning, which addresses the challenge of device heterogeneity, non-IID environments show negative effects due to the frequent updates after each client causes a global model update. Gajanin [Gaj24] extended the Flower framework to implement asynchronous federated learning and provided the work as an open-source implementation on Github.

Doriguzzi et al.[DCS24] propose a framework called FLAD (adaptive Federated Leaning Approach to DDoS attack detection), providing an enhancement of FedAvg for cybersecurity application in non-IID environments with highly sensitive information. It allows assigning more computation to members whose attack profiles are harder to learn without sharing test data with a central entity. Additionally, their system allows for dynamic scenarios, in which new attack profiles must be taken into account in the trained models. Doriguzzi et al.[DCS24] show that FLAD offers better performance in terms of convergence time and accuracy. FLAD is validated using CIC-DDoS2019, a dataset of network activity with 13 types of DDoS attacks, provided by the University of New Brunswick [DCS24].

The work by Aouedi et al. [APMS22] proposes a federated semi-supervised learning scheme to ensure that local private data is not sent to a remote entity while also detecting attacks with a high accuracy. For their work, two IIoT datsets are used, including a popular benchmark dataset for security research with data from a gas pipeline. However, their approach also does not account for any device heterogeneity, as it is also run on a single machine and utilizes the synchronous model aggregation algorithm FedAvg, making it prone to stragglers when applied in a more heterogeneous environment.

A systematic survey regarding federated learning in edge computing was conducted by Abreha et al. [AHS22], describing the current state of federated learning in edge computing, recent advancements, open problems and also addressing research being done in the field of heterogeneity management and future research directions. It shows that there already exist potential solutions to device heterogeneity in federated learning which can potentially be employed in future research [AHS22]. For example, FedProx [LSZ+20] was proposed as a variation of FedAvg, which associates tasks with the available resources on each device based on the work that needs to be performed. In that way, FedProx provides a solution to the device heterogeneity challenge while still running synchronously [AHS22, LSZ+20].

Xu et al. [XQXG23] conducted a survey specifically focusing federated learning in heterogeneous device environments in an asynchronous manner. They show that synchronous aggregation methods in synchronized federated learning settings encounter resource utilization limitations due to the need to wait for slow devices (stragglers). It summarizes existing asynchronous federated learning approaches not only regarding device heterogeneity, but also data heterogeneity, security and privacy. One key conclusion is that there currently do not exist sufficient real-life evaluation testbeds for asynchronous federated learning using heterogeneous devices [XQXG23].

CHAPTER 4

Architecture Design

In this chapter, we first define a structured process for the architectural transition from a homogeneous federated learning system architecture to a heterogeneous environment based on the literature review conducted in chapter 2. We then review the baseline system with respect to relevant architectural design aspects and patterns, with the focus on heterogeneous device environments and DoS attack mitigation. Based on our findings, we propose the architectural design of the improved framework, introducing a uniform system deployment on heterogeneous edge devices, asynchronous model aggregation, dynamic data utilization, backup mechanisms, and centralized model evaluation.

4.1 Transition From Homogeneous to Heterogeneous System Architectures

Based on the literature review, we define the following methodology for transitioning a homogeneous federated system architecture to a heterogeneous environment:

1. **Review** the baseline regarding relevant patterns and design aspects
2. **Show** potential limitations, drawbacks, and trade-offs
3. **Apply** derived modifications
4. **Evaluate** the improved framework against the baseline

Step 1 aims to gain a deeper understanding of the system architecture based on relevant design aspects and patterns found in research. The goal of *Step 2* is to find limitations, drawbacks, and trade-offs with respect to the underlying environment and use case. In *Step 3* proposed solutions are applied to the baseline, aiming to provide an improved

framework mitigating previous limitations regarding the heterogeneity. The purpose of *Step 4* is to empirically evaluate the improved framework against the baseline using key metrics and justify, as well as validate the design choices. This structured process aims to guide us in the design and evaluation of our work.

4.2 Baseline Architecture Review

The work of Bekbulatova et al. [BMD23], as described in chapter 3, compares semi-supervised network anomaly detection in traditional centralized learning to a federated setting. The framework can be classified as horizontal FL and the general architecture can be categorized into a centralized, client-server approach. This is the standard approach achieved with the federated learning framework Flower [BTM+20]. The framework consists of a server and multiple clients, where each client consists of a *SuperNode* and a *ClientApp*, and the server consists of a *SuperLink* and *ServerApp*. *SuperNodes* and *SuperLink* are long-running processes responsible for communication between the federation. The *SuperLink* is a process on the server side that forwards task instructions to *SuperNodes* and receives task results from *SuperNodes*. *SuperNodes* are processes that connect to the *SuperLink*, asking for tasks, executing tasks, and returning results to the *SuperLink*. The actual project-specific code lies in the *ServerApp* and *ClientApp*, both short-lived processes, containing all custom aspects of a federated system, such as client selection and configuration, result aggregation, local model training and evaluation, as well as pre- and post-processing. Flower also allows for multi-tenancy, meaning each client can have multiple *ClientApps* and the server can have multiple *ServerApps*, where depending on the run, different setups can be used in the same federation. The work by Bekbulatova et al. makes use of the basic Flower setup, without multi-tenancy.

The server in Flower additionally acts as a client registry (see section 2.4.2), as it needs information on all connected devices. As all available Flower aggregation strategies are synchronous (at least at the time of conducting this work), the Flower server only aggregates the results after receiving the result of each individual participating client. The aggregation model used in the work of Bekbulatova et al. [BMD23] is FedAvg, the most popular aggregation approach found in recent research, which has been described in section 2.4.

The federated setup for the experiments in the baseline framework consists of five clients and one server, running locally on a single machine. A visualization of this setup can be seen in 4.1. All clients in this IIoT scenario utilize the same model configuration, where each client has both labeled and unlabeled data and an equal subset of the IIoT dataset, as well as a roughly same proportion of labeled and unlabeled data. Since all clients run on a single machine, this framework does not account for any device heterogeneity. All clients finish their tasks in roughly the same amount of time, making the synchronous aggregation algorithm FedAvg seem very efficient without any idle times on the clients. However, in a real federated setting, devices are not homogeneous and have different computational power, from GPU and CPU capabilities, to memory

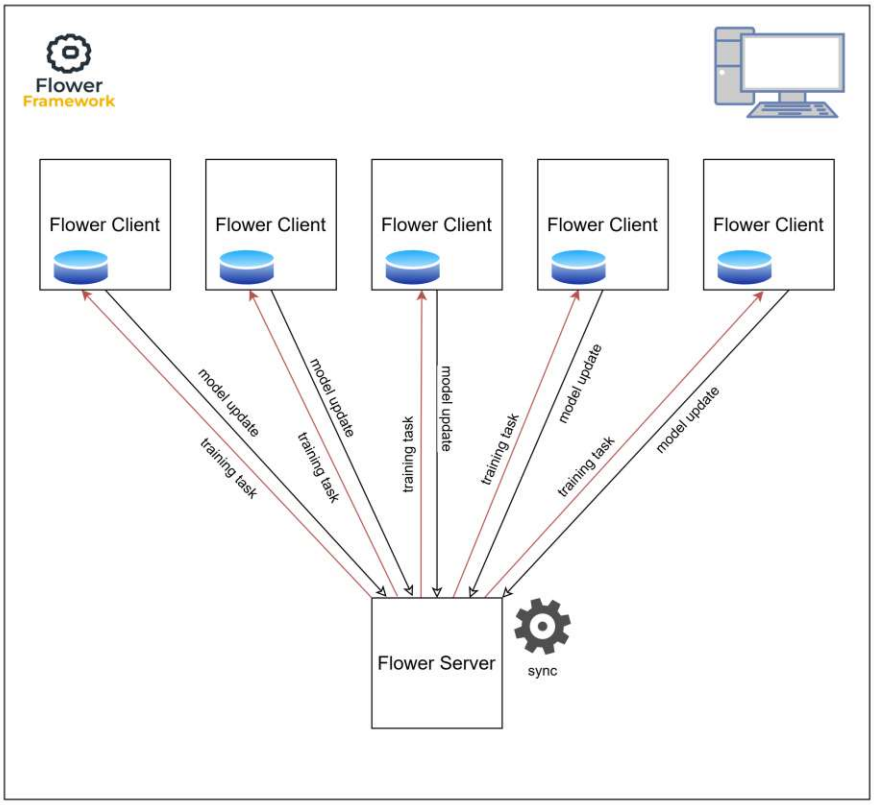

Figure 4.1: Baseline architecture as used by [BMD23].

storage and network connection, which ultimately leads to varying task execution times between clients. Handling heterogeneous clients is a crucial part of edge computing and federated learning, where edge devices range from small sensors to mobile phones and general-purpose computers.

Although possible within Flower, the baseline framework does not use any advanced client selection algorithm. The number of clients is set initially when the server is created, and all clients are receiving training and evaluation tasks on each round. The server does not start distributing training tasks until the set number of clients are registered. Furthermore, the server first initialized the global model with randomized weights. While a training round is ongoing, the server waits until all clients respond with results. After receiving all results, it distributes the new averaged global model and initiates the evaluation, which are tasks sent to all clients to start the evaluation. Each client then tests the current local model on its local dataset, specifically on the test split. The split of the dataset in the baseline framework is 60%/40% between train and test data. The evaluation in the baseline produces ten metrics on the model. The main metrics used for evaluating the proposed framework are AUC-PR and AUC-ROC. AUC-PR is a metric for calculating the area under the Precision-Recall (PR) curve, while AUC-ROC is used

to calculate the area under the Receiver Operating Characteristic (ROC) [BMD23].

Although anomaly detection with respect to DoS attacks is shown to be very accurate in the works of Bekbulatova et al.[BMD23], the proposed system is not able to handle actual DoS attacks on the federated environment itself due to two limitations. First, the synchronous aggregation, as it is currently implemented, does not allow for any failing nodes. As soon as a device shuts off from the federation, the system will be in an error state and entirely blocking the federated operation, because the server needs all clients to be available for a training round and aggregates the results only when all clients returned their respective results. Additionally, since DoS attacks usually heavily impair and slow down single devices, in a situation where the device does not completely fail, it still slows down the aggregation due to the server waiting for all client results. Especially in a zero-day-attack, time is crucial in order to find and respond to attacks in minimal time.

The second limitation of the proposed system is lost data in case of client failure due to DoS attacks. Each device has its respective local dataset on which training and evaluation are performed. This is generally the case with federated learning, but in a scenario in which each client's data is essential and needed for an accurate global model, losing the dataset of a device in case of failure might not be desirable. In a DoS aware environment, finding a solution which still allows the use of the client's data despite being attacked would enhance the resilience of the system.

The current system also does not support any new client data. The Flower Client is instantiated with the dataset at startup and even though the underlying tables can change, any new data is disregarded and is not loaded into the Flower Client for its duration. In an environment where it is crucial to respond to new attacks as soon as possible, feeding new data into the clients and adapting the model to changing attack schemes, making use of newly available data is essential.

4.3 Architecture Design of the Heterogeneous Framework

In this section we are proposing an architecture counteracting some of the limitations we discussed in the previous baseline review. We focus our improvement efforts on the heterogeneity of devices and on effectively mitigating DoS attacks on the federated system. Figure 4.2 shows a visualization of our proposed architecture design.

In order to further evaluate our system regarding the heterogeneity of devices, we set up a real-life testbed in a physical network consisting of three devices with varying computational capabilities. We argue that three different devices are sufficient to show the limitations of using a synchronous aggregator. To gain further insights, we would need to scale the number of devices in the testbed to a degree that is not feasible for this work. For example, since only a single server is involved, having thousands of devices simultaneously communicating with the server creates a bottleneck that can lead to performance issues or complete failure of the system. In systems with a high number of devices, hierarchical or regional architectures can be employed to counteract this

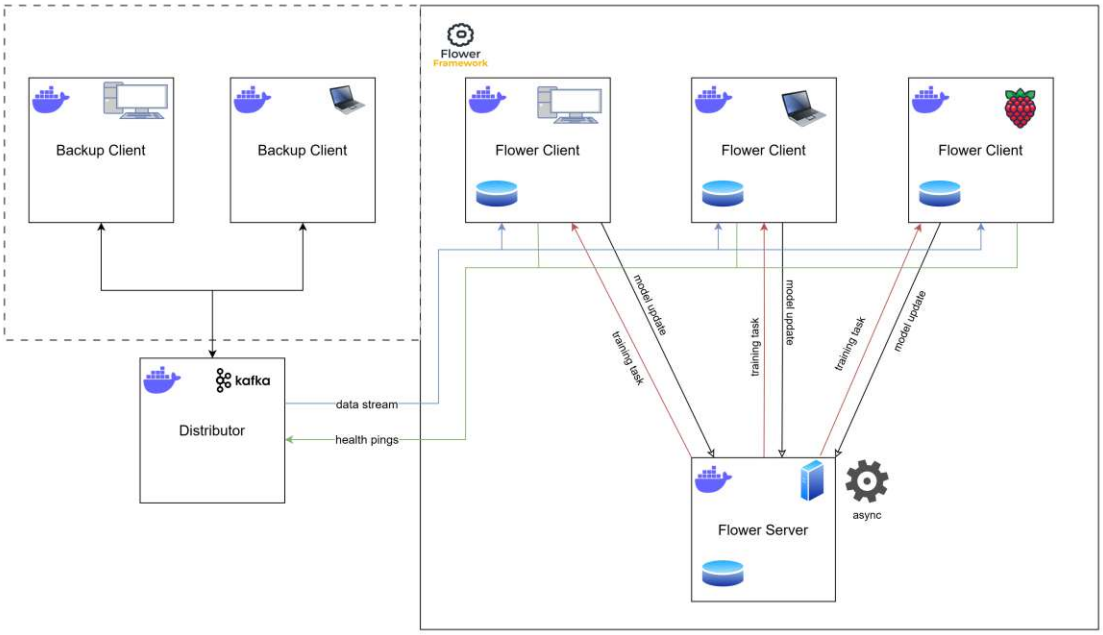

Figure 4.2: Architecture design of our heterogeneous framework.

challenge. As this is not the focus of this work, we will continue with a single centralized server as used by Bekbulatova et al. [BMD23], but suggest this area as a subject for future research.

Because manually installing the required dependencies on multiple devices with different operating systems is cumbersome, the second improvement we propose is to "dockerize" the system. Docker[1] makes use of operating-system-level virtualization and is a popular and widely-used product in software development to package software and run them in so-called containers, regardless of the host system. The containers include everything the software needs to run, thus no manual installation of dependencies is needed on each client, which is especially useful when deploying software to heterogeneous environments.

To counteract inefficiencies regarding stragglers and the unreliability of nodes with synchronous aggregation, we propose the use of an asynchronous aggregator on the server. Asynchronous federated learning is not yet widely adopted, and federated learning frameworks such as Flower do not support asynchronous aggregation out of the box. However, we argue that it is a vital part in heterogeneous environments due to its possibilities to handle stragglers and deal with failing nodes. During a DoS attack, in which there is a high chance that nodes become unavailable, it is crucial to employ a system that is resilient enough to handle failing nodes.

We argue that in a real-life, industrial IoT setting, devices producing data, such as small sensors, are often not capable of training computationally intensive machine learning

[1] https://www.docker.com/

models. Additionally, it is crucial to adapt to changing environments and training models on new data allows for quick adaptations and responses when confronted with zero-day attacks and changing attack schemes. To accommodate this, we separate the data creation and data storage on the client devices. Federated clients are still working on their local dataset, but the production of data might be on separate devices. We introduce the *Distributor*, a separate entity acting as the data producer for all clients, as seen in Figure 4.2. To allow clients to receive data in real-time, we make use of the open-source event streaming platform Kafka[2], providing a high-throughput, low-latency solution for real-time data. Kafka allows for subscribing and publishing to events through so-called topics. On startup, federated clients connect to the distributor, which then in turn creates a new topic for this client. The distributor streams new data on each client topic, and the clients update their local dataset with the received data.

In order to handle DoS attacks more effectively and the fact that through our previous changes we separated the creation of data from local datasets, we further propose a self-healing system in case of node failures through backup clients. A backup client can be started as soon as the system detects that a client in the federation fails to operate in an acceptable manner. To achieve this, we further utilize the before introduced *Distributor*. Additionally, it keeps track of all connected devices in the federation and reacts quickly, if a failure is detected. All clients and backups initially connect to the *Distributor* at startup and send health pings in a set interval. The *Distributor* saves the last health ping received from each client, and in case a client does not send an additional health ping in a pre-defined time span, it will initiate the backup plan. The backup client then acts as a Flower Client and replaces the failing node. Through our data separation step as mentioned before, backup clients connect to the data stream of the replaced client and save it to their local dataset. Local datasets could further be replicated in a distributed manner to prevent data loss from failing nodes, but for our work, we simulate this behavior by providing initial datasets to each client and backup.

Figure 4.3, shows a flow chart of all entities interacting with our proposed *Distributor*. The *Distributor* first creates the distributor topic, on which backups and client register. After a backup registers with its generated id, the *Distributor* saves that for future communication. After a client registers, the *Distributor* creates a client specific topic, on which it will stream new data and create the first health entry. It spawns a new data thread if it does not already exist and sends batches of data on the client topic in predefined intervals. Additionally, the *Distributor* checks the last health ping of each client to determine when a node is failing. If the time span exceeds a predefined threshold, it will initiate the backup using the first available backup, then removing it from the backup list. Finally, it stops the data thread for the failed client. After registering on the distributor topic, the client has two repeating tasks. First, it listens on the client-specific topic for new data and appends received data to the local dataset and secondly, it sends a health ping on the distributor topic in a predefined interval. A backup registers on the distributor topic and waits until a backup message is received. After receiving a message,

[2]https://kafka.apache.org/

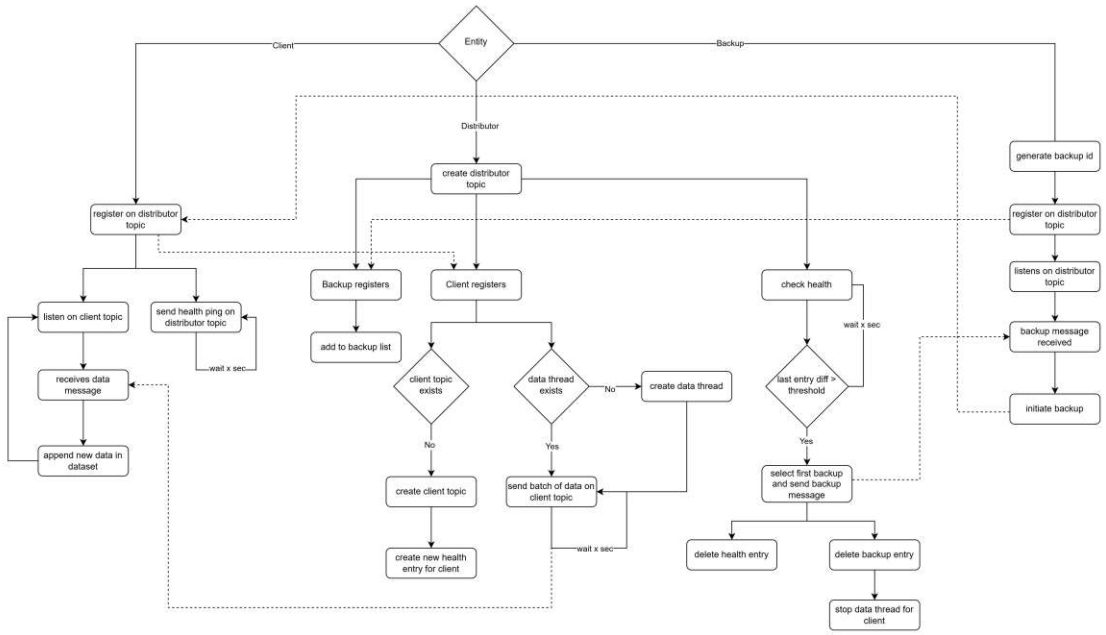

Figure 4.3: Flow chart of entities interacting with the Distributor.

it will initiate the backup, meaning running the client program. After that, it acts as an additional client in the federated environment.

To show how backups are initiated, Figure 4.4 visualizes this scenario. In this example, *Client 1* first registers with both *Server* and *Distributor*, and receives new data on its client specific topic and regularly sends health pings. It receives a training message from the *Server* and starts training. During training, it becomes unavailable (e.g. because of a DoS attack). After some time, the health check threshold is exceeded for this client, thus the *Distributor* initiates a backup by sending a backup message with the client ID, 1 in this case, since *Client 1* failed. The backup starts the client program and connects to the *Server* and *Distributor*, and after that acts as a healthy *Client 1*.

The final improvement we propose in this work is an additional centralized evaluation. This change is mostly motivated by the asynchronous aggregator, which allows the server to get an accurate representation of the global model after each client update. However, this approach does offer additional possibilities in aggregation improvements. For example, having a local validation dataset, which is not part of any client data, could be used to determine whether a client update improves the global model. If it increases the model accuracy, it is embedded in the global model, otherwise it is discarded. As implementing such behavior would exceed the scope of this project, it will not be further discussed, but it will be listed for future research.

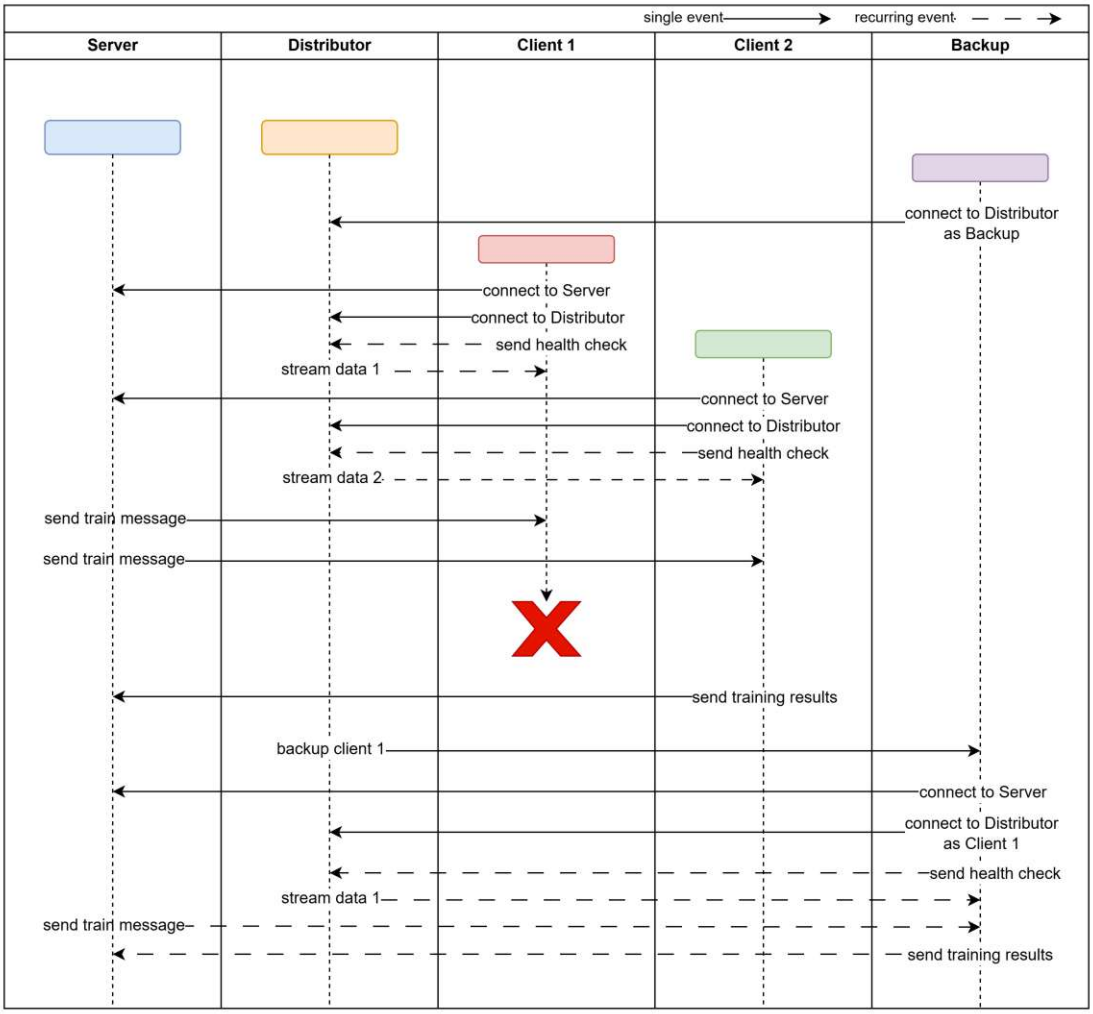

Figure 4.4: Sequence diagram of backup initiation.

CHAPTER 5

Implementation

In this chapter we describe and discuss aspects of our framework implementation. The architecture design, as described in the previous chapter, is the basis of our implementation approach. We discuss the introduction of the uniform system deployment, the integration of an asynchronous model aggregation within Flower and our approach to implement dynamic data handling and a backup mechanism in our framework. Our implementation is available as an open-source repository on Github[1].

5.1 Dependency Upgrades and Uniform System Deployment

The first step of our implementation process consists of updating the framework regarding its dependencies, including the upgrade of the python version to `3.12.2` and the Flower version to `1.14.0`, resulting in minor changes to the client instantiation. In order to facilitate the deployment of our system on multiple heterogeneous devices with different operating systems, eliminating the need to manually install required dependencies on each device was the logical next step. To "dockerize" our system, we used a base image containing the `conda`[2] package manager, due to a few complications with the dependency installation using `pip`[3]. In our extended image, necessary dependencies are installed and entry point scripts are provided to start the nodes upon container creation. Additionally, we provide `docker-compose`[4] files for various setup configurations within the testbed, which results in a single command needed for our experiments.

[1] https://github.com/Riesal11/Deep-SAD-FL
[2] https://anaconda.org/anaconda/conda
[3] https://pypi.org/project/pip/
[4] https://docs.docker.com/compose/

5.2 Asynchronous Model Aggregation

Although the Flower framework is, as of conducting this thesis, bound to synchronous model aggregation methods and does not support asynchronous aggregation, its underlying architecture allows for extensive customization. To this end, it is possible to create custom `Server`, `Strategy`, `ClientManager`, and `History` classes to implement an asynchronous approach within the `Flower Server`. In this work, Gajanin [Gaj24] adapted the `Flower Server` to support asynchronous federated learning and evaluated their work in the context of human activity recognition within non-IID data environments. His implementation of the `Strategy` is based on Fedasync[XKG19], which builds on `mixing alpha` as a parameter while aggregating the models, dictating the weight each model (global vs. new local model) holds. This parameter is then multiplied by the sample proportion of the client to normalize the magnitude of the update [Gaj24]. As this implementation by Gajanin [Gaj24] was published as an open-source implementation and is currently under review for integration within the Flower Framework, we utilized and integrated this approach for our use case and thus, made no changes to the `Strategy`. Our main adaptations after integrating this asynchronous approach in our framework referred to the `Server` to allow new registered clients to start training during the federated learning process, fostering our changes regarding DoS attack mitigation and system resilience. In contrast to the implementation of Gajanin [Gaj24], our data configuration takes place at the client side and we removed server-side data handling of the clients. We made further changes regarding the centralized (server-side) evaluation, which initially evaluated the global model periodically, to now evaluate upon the reception of model updates from clients.

5.3 Dynamic Data Handling and Backup Mechanism

Adapting to changing environments and training models on new data is crucial when confronted with zero-day-attacks. Based on our design decisions to separate data creation and data storage on client devices, we identified Apache Kafka[5] to accommodate this decision through the introduction of event streams. An edge node where data is created can connect to a topic specific to a `Flower Client`, where it then publishes new data. The `Flower Client` subscribes to its topic and continuously appends the received data to the local dataset. We simulate this behavior by introducing the `Distributor`. The `Distributor` leans on the idea of a client registry and a client selector and is a process that manages active clients and initiates the backup mechanism. On startup, it creates a general `distributor_topic` on which the communication between the `Distributor` and clients take place, and two `Threads`, the listener for the `distributor_topic` and a `HealthThread`. It registers itself on this topic through a `KafkaConsumer`, while a client registers as a `KafkaProducer`. In Kafka, messages must be of type `byte`, thus we use `binascii`[6] for our (de)serializers. Whenever a client registers, a new topic

[5] https://kafka.apache.org/
[6] https://docs.python.org/3/library/binascii.html

Package	Version
flower	1.14.0
torch	2.4.1
torchvision	0.19.1
numpy	1.26.4
pandas	2.2.3
scikit-learn	1.6.1
kafka-python-ng	2.2.2

Table 5.1: Core python packages and respective version numbers used in our implementation.

`client_$CLIENTID_topic` and a `DataThread` is created for the simulation of new data. Then it periodically sends a batch of data from a client-specific local dataset through a `KafkaProducer` on the respective client topic, which in our experiment was set to 100 new messages every 5 seconds. In order to split the full dataset accounting now for subsets which will be streamed during the experiment, we created a `DataPartitioner`, that, depending on the number of clients and parameters such as `stream_split_fraction` and `server_validation_fraction`, produces the local client datasets, respective datasets `Distributor`, in addition to the server validation set. In our current setup, the backup clients also initially contain a local dataset for the sake of easier evaluation. On startup, a backup client connects to the distributor by generating an ID and starts a `PollingThread` to wait for a backup request. When receiving a backup request with a client ID, it executes the client program and acts as a regular client, including connecting to the respective client topic. The `Distributor` has a dictionary for the health check, consisting of client ID and the last health ping received from that client. Values get updated through initial registration of clients and a regular messages on the `distributor_topic`, where `health` is the key and the respective client ID the value of the message.

For clarity and reproducibility, we used python `3.12.2` for our framework implementation. The main python packages, alongside their version numbers, are summarized in Table 5.1.

CHAPTER 6

Evaluation

In this chapter, we evaluate our proposed framework against the baseline. First, we define the metrics we use to effectively compare the two frameworks. Then we describe the environment in which the experiments are run and present the evaluation of our baseline as well as the evaluation of our improved framework. Finally, we summarize the results and takeaways.

6.1 Quantitative Performance Metrics

We use **AUC-PR** and **AUC-ROC** as model performance metrics, as these were the main metrics used to evaluate the baseline framework by Bekbulatova et al. [BMD23]. AUC-PR represents the area under the Precision-Recall (PR) curve, while AUC-ROC is a metric that constitutes the area under the Receiver Operating Characteristics (ROC) curve [BMD23]. The ROC curve is a graph that illustrates the performance by plotting true positive rate against the false positive rate at each threshold, and through calculating the Area Under Curve (AUC) it is possible to quantitatively compare different models [KGG14]. The PR curve, on the other hand, plots the precision (positive predictive values) against the recall (true positive rate).

In addition to AUC-PR and AUC-ROC, we include two more metrics for evaluating the model performance, the **F1-score of the normal class** and the **F1-score of the anomalous class**. The main reason for this decision, besides its popularity, is to gain further insights on the normal, non-anomalous class and the anomalous class, especially when comparing the frameworks regarding DoS and other attacks. The F1-score is defined as the harmonic mean of precision and recall [B$^+$19].

The main metrics we identified for the comparison of the aggregation methods between the baseline and our proposed framework regarding the heterogeneity of the client devices are the **percentage idle time** and the **update rate** of the clients. We define idle time as

the time not spent in a working or failure state (startup, training, testing, disconnected) as a percentage of the total active time. We further define the update rate as total model updates received by the server per straggler update, where the straggler is the slowest client device in the federation. As both idle time and update rate are dependent on dataset size, our refined definitions allow us to compare the results more accurately across environments. For our proposed framework, we additionally introduce the **backup time**, which we define as the time it takes the system until a backup is ready for training, starting from the moment the disconnect of a client occurs.

6.2 Experiment Environment

The devices used for the following experiments consist of one desktop computer, one laptop, and one Raspberry Pi 4 Model B. The desktop computer runs on a Windows 11 Home OS, an AMD Ryzen 7 3700X @ 3.60GHz CPU, a NVIDIA GeForce RTX 3060 Ti GPU, and 16 GB of RAM. The laptop runs on a Windows 10 Pro OS, an Intel Core i5-7200U @ 2.50GHz CPU, an Intel HD Graphics 620 GPU, and 16 GB of RAM. The Raspberry Pi 4 Model B runs on Raspberry Pi OS, a Cortex-A72 @ 1.80GHz CPU, no GPU, and 4 GB of RAM. All devices are connected to the same local network.

All of our experiments use the CPU as our computation device in federated learning and we run all entities in Docker containers, contrary to the experiments conducted by Bekbulatova et al. [BMD23]. The main reason for using CPU as the computation device is being able to run a federated client on devices without a GPU, such as a Raspberry Pi. This makes the system more accessible in IIoT environments if the results show acceptable performances. Using a Docker environment in which our system is run decouples it from the actual host setup, such as operating system, and lets us deploy our system on different host environments without the need of manual setups on each device. Additionally, we use three client devices in each experiment, while Bekbulatova et al. used five. When running an experiment locally, we always use the desktop computer for all three clients. For experiments in the heterogeneous testbed, *Client 1* always represents the desktop computer, *Client 2* the laptop and *Client 3* corresponds to the Raspberry Pi. The underlying dataset for the experiments is the same as in the baseline framework, the WUSTL-IIOT-2021 dataset [M. 21]. Each training round consists of 50 epochs.

6.3 Baseline Framework Evaluation

To evaluate our work, we first establish the baseline since our setup differs from the one used by Bekbulatova et al.[BMD23]. To do that we compare a local experiment against an experiment in our heterogeneous testbed. For each experiment we run it once with the anomalous class consisting only of DoS attacks and once with the anomalous class consisting of all attacks (DoS, Command Injection, Backdoor, Reconnaissance). Model performance results are visualized in Figure 6.1, where blue colors indicate the local setup with only DoS attacks, red colors the federated setup with only DoS attacks,

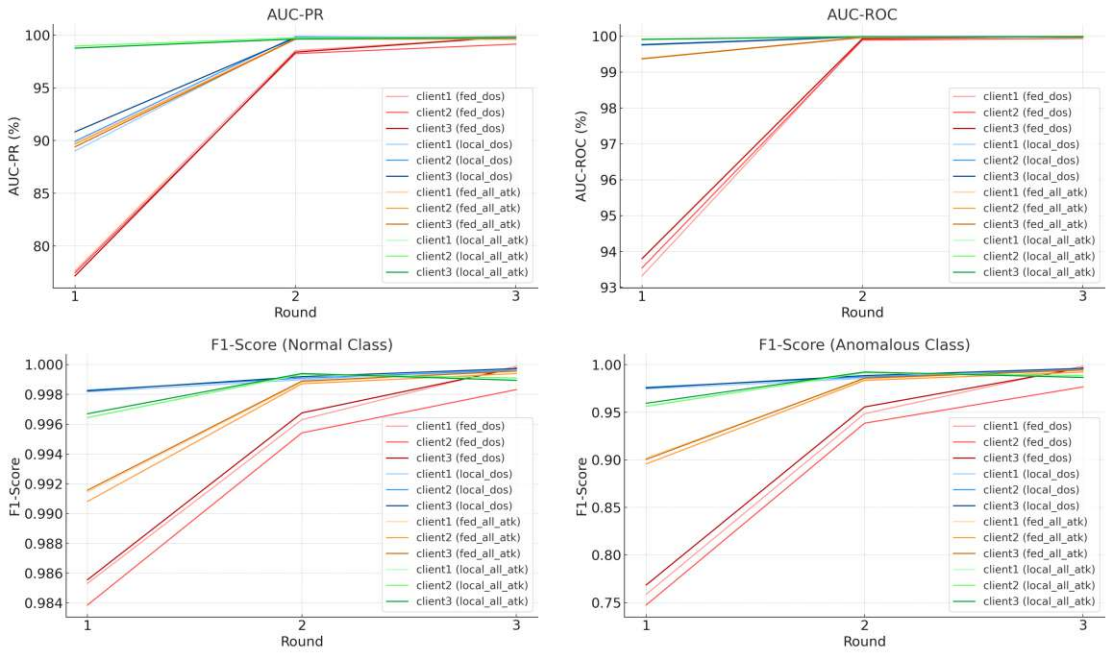

Figure 6.1: Visualization of the model performance of the baseline framework in a local setup (local_*) and a setup using our heterogeneous testbed (fed_*), including results from training exclusively on DoS attacks (*_dos) and training on all types of attacks (*_all_atk).

green colors the local setup with all types of attacks and the yellow colors the federated setup with all types of attacks. When comparing local_dos to fed_dos, we can see that in the first round, the federated setup performs worse than the local setup across all performance metrics. The most notable differences are at the AUC-PR and the F1-score of the anomalous class, where values between 0.74 and 0.77 are recorded. Although the F1-score of the normal class is worse in fed_dos compared to the other setups, it still shows very high values (0.984) in the first round. While still showing differences, all setups have an increased model performance in round 2 and by round 3, all setups seem to be converging, showing exceptional results. Overall, the setups including all attacks (local_all_atk and fed_all_atk) show better model performances than their DoS counterparts in most cases, suggesting that the models are very good at identifying the additional attacks as anomalous even after a single round of training. However, these results do not show how good the model is at identifying specific attacks, since we still use binary classification between the normal and anomalous class.

In addition to the model performance, we evaluate the baseline regarding its activity times in a heterogeneous environment. We run the baseline in our testbed with two dataset sizes. In the first experiment, we use the full dataset at each client, and in the second we spread the dataset across clients so each holds a third of the total dataset. We measured average and total training, testing, and idle times as well as the percentage idle

Full dataset	Client 1	Client 2	Client 3
Average training time per round (s)	589.45	1004.01	4570.37
Average testing time per round (s)	7.25	8.63	44.42
Average idle time per round (s)	4010.38	3609.53	0.21
Total training time (s)	1768.35	3012.04	13711.12
Total testing time (s)	21.75	25.89	133.26
Total idle time (s)	12031.14	10828.59	1.09
Percentage idle time (%)	85.63	77.07	0.01

Table 6.1: Statistics of activity periods of the baseline framework in our heterogeneous testbed using the full dataset at each client.

Reduced dataset	Client 1	Client 2	Client 3
Average training time per round (s)	223.54	323.69	1491.79
Average testing time per round (s)	2.05	2.74	14.61
Average idle time per round (s)	770.68	709.94	0.09
Total training time (s)	670.63	971.06	4475.38
Total testing time (s)	6.15	8.22	43.82
Total idle time (s)	3853.39	3549.69	0.45
Percentage idle time (%)	80.30	73.99	0.01

Table 6.2: Statistics of activity periods of the baseline framework in our heterogeneous testbed using a reduced, exclusive dataset at each client, corresponding to a third of the total dataset.

time. The results are shown in Table 6.1 and Table 6.2. In the case of the full dataset, the total time for the three training rounds was 14048 seconds (3.90 hours). *Client 3* needs 4570 seconds on average for a training round consisting of 50 training epochs, while it needs 44 seconds for testing and has almost no idle time. *Client 1* on the other hand needs 589 seconds for a training round and is idle for 4010 seconds each round, totaling an idle time of 12031 seconds over three rounds and resulting in a percentage idle time of 85% (3.34 hours). As expected, the results for the reduced dataset are lower and we see similar but slightly lower values for the percentage idle time compared to the previous case. The server receives 1 update from each client until *Client 3* finishes, thus resulting in an update rate of 3.

Figure 6.2 shows a visual representation of activity periods for each client, as well as a zoomed-in version of the first round. We can see that the startup time using the full dataset is 176 seconds. The server initiates the federated training round as soon as the three clients are connected and *Client 1* already finishes its training task after 773 seconds. *Client 2* finishes soon after (1211 seconds), but both have to wait until *Client 3* finishes its round (after 4783 seconds) until the server receives all updates and initiates the testing task. The testing periods for *Client 1* and *Client 2* are not properly visible in the figure due to their short duration, but they always occur at the same time as

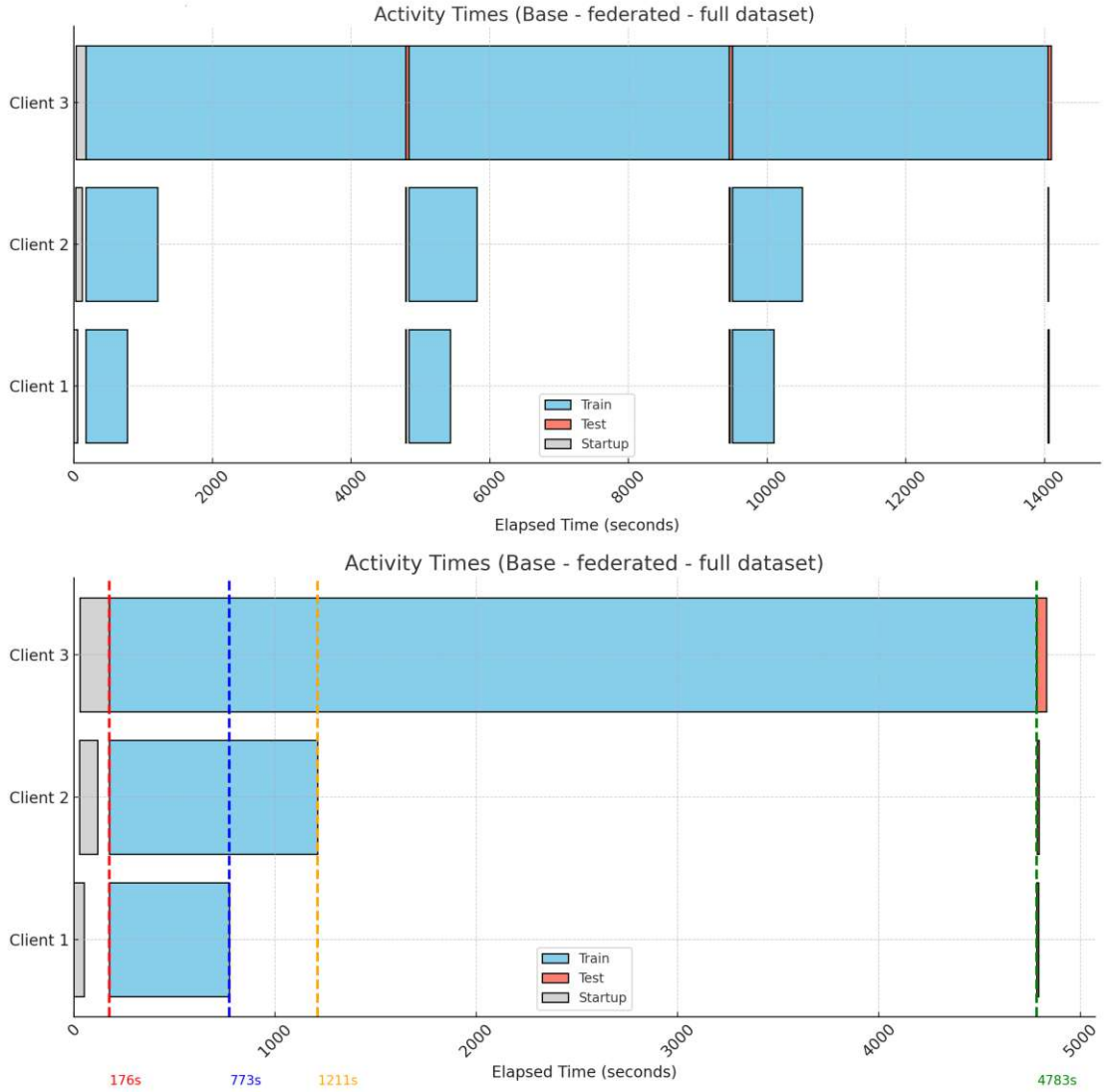

Figure 6.2: Visualization of the activity periods (Startup, Train, Test) of the baseline framework using the full dataset at each client, while highlighting idle periods in heterogeneous environments. The figure on top shows the timeline for the full experiment (14048 seconds), the bottom figure shows a zoomed-in version of the first round (4828 seconds), including training start and end time marks for each client.

Client 3. Figure 6.3 shows the activity periods for each client using the reduced dataset, resulting in a total runtime of 4797 seconds. From these results, we can clearly observe that in heterogeneous environments, the slowest device slows down the entire system in the baseline framework, resulting in high idle times for devices with more computing resources.

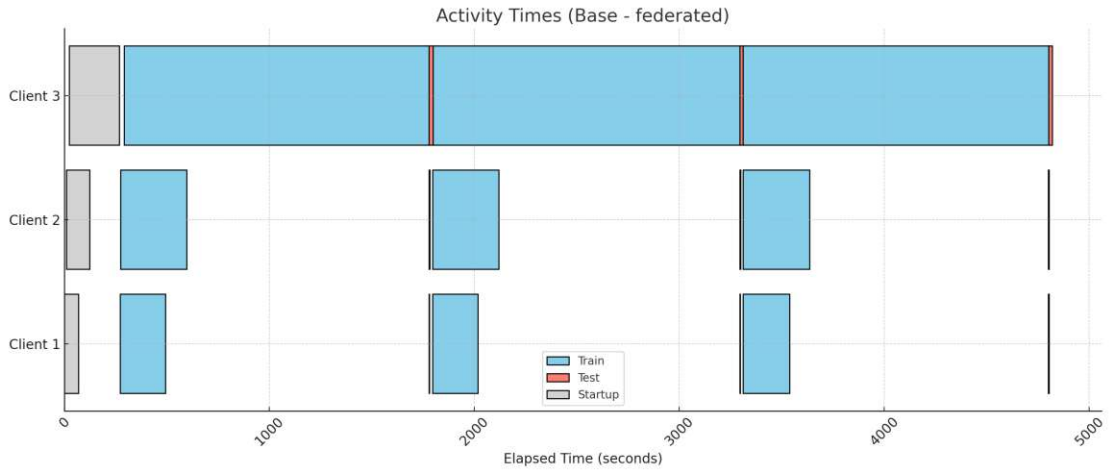

Figure 6.3: Visualization of the activity periods (Startup, Train, Test) of the baseline framework using the reduced, exclusive dataset at each client. The total runtime of the experiment amounts to 4797 seconds.

6.4 Heterogeneous Framework Evaluation

For the evaluation of our proposed framework we adjusted the dataset sizes due to the additional entities requiring a split of the dataset. An exclusive portion of the data resides at the server, as well as a portion of each client's data resides at the distributor for data streaming. The total client dataset is evenly split among the clients. We use a server validation fraction of 0.2 of the total dataset and a stream split fraction of 0.2 of each client's data. This results in an individual local client set fraction of 0.192. Approximately at the 1000 second mark, *Client 2* is shut down, which initiates the backup behavior.

In addition to the federated evaluation as seen in 6.1, for our proposed framework evaluation we include a centralized evaluation at the server. As mentioned before, the server now possesses an exclusive dataset, which is not used by any clients, and performs an evaluation after each model update. Figure 6.4 visualizes the results of the model performance of our proposed framework. The federated evaluation shows exceptional results across all four metrics even after a single round of training. The lowest score for AUC-PR is 99.63% (*Client 3* Round 2) and the highest score is 99.98% (*Client 1* Round 1). In the case of AUC-ROC, the lowest recorded score is 99.98% (*Client 1* Round 5), while the highest score, achieved by multiple clients, is 100.00%. The recorded F1-scores of the normal class range from 0.99970 (*Client 2* Round 1) to 0.99988 (*Client 1* Round 4). The lowest score for the F1-score of the anomalous class is 0.9958 (*Client 2* Round 1) and the highest score is 0.9983 (*Client 1* Round 4). These results show that within our environment, our proposed asynchronous setup not only converges faster, but also performs better than the synchronous baseline setups. The centralized (server-side) evaluation also shows very good results, but the model does not seem to be as consistent

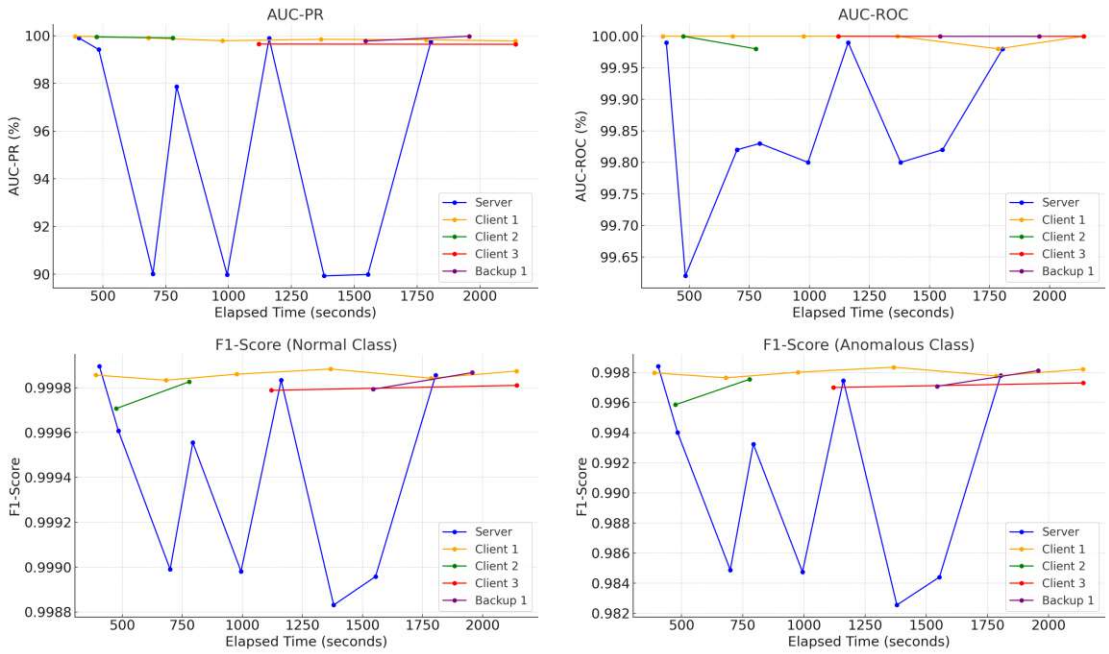

Figure 6.4: Visualization of the model performance of our proposed asynchronous framework in our heterogeneous testbed, including a centralized (server-side) and federated (client-side) evaluation approach.

as the federated (client-side) evaluations. The results vary between resembling the same performance as obtained in the federated evaluation and being slightly lower. The most noticeable differences can be observed in the AUC-PR graph, where the results range from 89.93% to 99.99%.

The update rate of our proposed framework cannot be easily compared to the baseline framework without some caveats. Since the dataset for each individual client is lower than in the baseline framework, the training periods are generally shorter, and therefore the model update rates are higher. Nevertheless, since the frameworks are run on the same resources, we can gain insights by comparing update rates relative to the bottleneck client (*Client 3*). While *Client 1* and *Client 2* send a single update to the server until *Client 1* sends an update using the synchronized aggregation method, *Client 1* sends three updates and *Client 2* sends two updates until the first update from *Client 3* is received. Between the first and second updates from *Client 3* we can observe the same behavior, although *Client 2* is replaced by a backup client in the process.

Figure 6.5 shows a visualization of the activity periods in the proposed framework using the asynchronous aggregation method. Idle times between training rounds are eliminated for *Client 1* and *Client 2*, allowing them to finish up to three training rounds and model updates in the time *Client 3* finishes one training round. After approximately 1000 seconds, *Client 2* simulates a disconnect from the federation. The federated system is

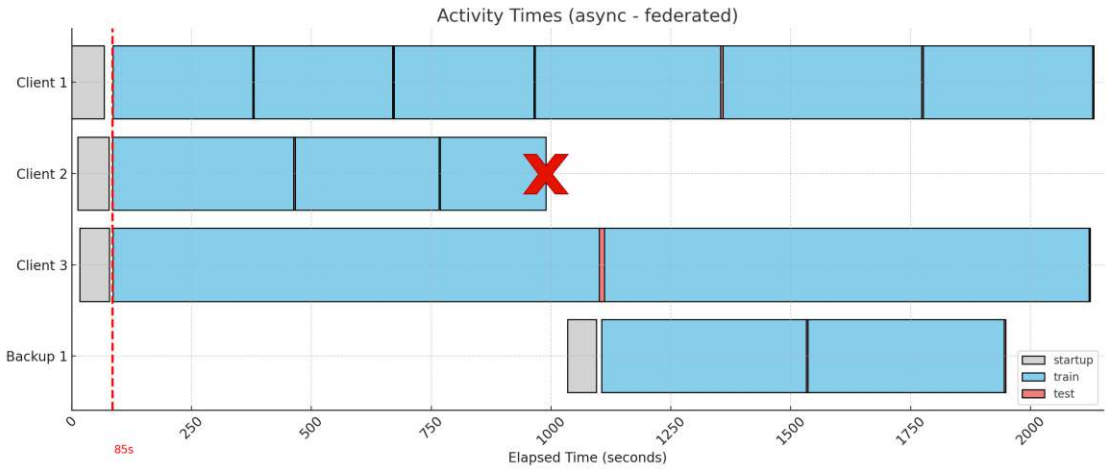

Figure 6.5: Visualization of the activity periods (Startup, Train, Test) in our proposed framework, highlighting the increased update rate in the asynchronous approach. *Client 2* disconnects after approximately 1000 seconds and is replaced by a backup client after 105 seconds.

unaffected by this incident and initiates the backup behavior after the last health check received from *Client 2* exceeds the threshold (30 seconds). *Backup 1* is selected to take its place and finishes two rounds of training until the system terminates. By the time *Client 3* finishes its first training round, the server received three updates from *Client 1* and two updates from *Client 2*, resulting in an update rate of 6. Similarly, the update rate at the mark of the second straggler update is also 6, despite the fact that *Client 2* disconnects and is replaced by a backup client. The time it took the system until the backup client was ready for training, previously defined as the backup time, was 105 seconds, of which 60 account for the start time of the client program.

6.5 Summary

We conducted an empirical evaluation of the baseline framework and the proposed framework regarding system and model performance in a small-scale, heterogeneous IIoT testbed. Based on Figure 6.1 we find that in the baseline framework, the local setup outperforms the federated setup across multiple model performance metrics in the first round and after three rounds, all setups show excellent results. Tables 6.1 and 6.2 provide statistics of the baseline regarding training, testing, and idle times of federated clients using different dataset sizes. We observe percentage idle times of up to 85.63%, showing that clients with more computing resources are heavily underutilized while having to wait for slow clients. Figures 6.2 and 6.3 visualize the activity periods for each client in the baseline framework, as well as a zoomed-in version of the first round. The results of the model performance of our proposed framework, as seen in Figure 6.4, show that within our environment, our proposed asynchronous framework not only converges

	Baseline Framework	Proposed Framework
Percentage Idle Time (%)	up to 85.63	up to 0.38
Update Rate	3	6
Backup Time (s)	not supported	105
System Deployment	manual	Docker container
Data Loading	at startup	before training
Dynamic Data	not supported	through data streams
Centralized Evaluation	not supported	after each client update
	local, fed	**fed eval, cen eval**
AUC-PR lowest (%)	89.02, 77.17	99.63, 89.93
AUC-PR highest (%)	99.90, 99.87	99.98, 99.91
AUC-ROC lowest (%)	99.75, 93.32	99.98, 99.62
AUC-ROC highest (%)	100.00, 100.00	100.00, 99.99
F1-Score (normal) lowest	0.9981, 0.9838	0.9997, 0.9988
F1-Score (normal) highest	0.9997, 0.9998	0.9998, 0.9998
F1-Score (anomalous) lowest	0.9746, 0.7475	0.9958, 0.9825
F1-Score (anomalous) highest	0.9962, 0.9985	0.9983, 0.9984

Table 6.3: Differences between baseline framework and proposed framework regarding system performance, supported features and model performance. For the baseline model performance, results from local_dos (local) and fed_dos (fed) are included. For the proposed framework, results from federated evaluation (fed eval) and centralized evaluation (cen eval) are included.

faster, but also performs better than the synchronous baseline setups using federated evaluation. We also highlight that there exist differences between federated (client-side) and centralized (server-side) model evaluation. Figure 6.5 visualizes the activity periods of the clients in our proposed framework using the asynchronous aggregation method and backup mechanism, showing the elimination of idle times, increased update rates and the resilience of the system in case of client failures, such as DoS attacks. We summarize the differences between the baseline framework and our proposed framework in Table 6.3.

CHAPTER 7

Limitations

In our framework, backup clients already contain a local dataset when they take the place of a failed client. Starting with an empty dataset would increase the time until the backup is at a model performance close to where the client was before failure. To this end, some recent amount of data could be stored for each client with the goal of keeping enough data in case a failure happens such that the backup has an acceptable initial model performance. However, this would potentially contradict the principle of federated learning, which builds on its privacy-preserving characteristics in the sense of utilizing local datasets on clients without sharing data with a central entity. In that case, a trade-off between a fully decentralized data storage and additional flexibility, availability, and resiliency of the system has to be made.

Although the outcomes of the evaluation show excellent results regarding the model performance of the framework, using a model that is capable of achieving close to 100% accuracy detecting DoS attacks after a single round makes the model evaluation challenging. Since machine learning models usually show some variance and there is not much improvement after each update, it is difficult to observe the impact of the increased update rate in our asynchronous federated learning approach. We included some evaluation of additional attacks, but it did not change the performance in significant magnitudes, perhaps partly due to still being a binary classification and DoS being the majority attack type of the dataset. Choosing a harder problem space for the model (e.g. multi-label classification or sophisticated attacks) potentially yields meaningful findings in that area.

Furthermore, a limitation of asynchronous federated learning, as employed in our framework, is that it shows a strong bias towards clients with shorter training times, as they contribute to global model updates much more frequently. This can either be due to computational resources, as in our case, or the size of the underlying client dataset. In heterogeneous and non-IID environments, this bias introduces a series of complications and necessary considerations, which have become a central focus of recent research.

CHAPTER 8

Conclusion

The evolution of Industry 4.0 has led to the emergence of highly interconnected devices and intelligent systems capable of advanced analytics and decision-making. Machine learning has become a key component of anomaly-based Intrusion Detection Systems (IDSs), while federated learning has emerged as a promising approach that enhances security and privacy. In Industrial Internet of Things (IIoT) environments, federated learning is leveraged to reduce communication overhead of sending massive amounts of heterogeneous data to a central server. Traditional approaches in federated learning, such as Federated Averaging (FedAvg), have been shown to face challenges in heterogeneous environments and Denial of Service (DoS) attack detection due to their synchronous model aggregation, leading to resource utilization limitation and a reduced convergence rate. Especially in zero-day-attacks, a short convergence time is essential to quickly distribute updated global models that include newly gained DoS attack information. Asynchronous federated learning emerges as a promising solution to address challenges in heterogeneous environments, particularly issues related to device unreliability, aggregation efficiency, and resource utilization. Nevertheless, current federated-learning-based anomaly detection approaches in IIoT environments fail to address device heterogeneity and system resiliency against DoS attacks.

In this thesis, we propose a novel framework for federated learning that optimizes the system architecture for heterogeneous environments, while focusing on DoS attack detection. To guide our transition from a homogeneous federated intrusion detection system architecture to a heterogeneous environment, we develop a methodology based on key architectural design principles and patterns, resulting in a structured process serving as the foundation of our framework development. Through this methodology, an existing federated-learning-based intrusion detection framework is established and reviewed, resulting in a comprehensive architectural design for our proposed framework, including a uniform system deployment on heterogeneous edge devices, asynchronous model aggregation, dynamic data utilization, backup mechanisms, and centralized model

evaluation. We then apply the changes according to our architecture design and develop a well-documented implementation to be published as an open-source framework. Finally, to demonstrate our improvements, we empirically evaluate the framework against the baseline in a small-scale, heterogeneous IIoT testbed with respect to system and model performance using key metrics.

The experimental results from the baseline framework show that devices with more computing resources are heavily underutilized, experiencing idle times up to 85% of the total experiment duration, highlighting the need for optimization of resource utilization in a heterogeneous environment. The introduction of asynchronous model aggregation eliminates idle times, and provides increased update rates, as well as resilience against DoS attacks. Furthermore, the proposed backup mechanism enhances DoS attack mitigation by enabling backup clients to promptly replace unresponsive clients. With respect to the model performance, our framework not only converges faster, but outperforms the baseline in AUC-PR, AUC-ROC and F1 scores in the federated evaluation.

CHAPTER 9

Future Work

We conclude this thesis by identifying potential avenues for future research that expand upon the proposed framework and related areas.

Our work utilizes a small-scale heterogeneous IIoT testbed consisting of three devices. In this scenario, communication with single entities, such as the `Server` and `Distributor`, does not lead to limitations, but could potentially become a bottleneck when scaling the number of clients in the federated environment. We suggest that this area be further researched and potential approaches explored to identify challenges in a more realistic deployment scenario in large-scale applications.

The introduction of centralized evaluation on the server yields additional possibilities in future evolutions of the framework. Approaches could be investigated that utilize the local validation dataset on the server to employ a precise model update selection process and determine the quality of individual client updates. Especially in the case of non-IID environments, this could potentially lead to significant improvements in model performance.

In our work, data streams are utilized for the inclusion of new data. Changing environments and capitalizing on new information is a central aspect of edge computing and federated learning. Potential avenues for future research include investigating the impact of stale data, determining the amount of data required for effective training, and addressing client drift in continuous federated learning. This architecture could further be used to increase the resilience of the system with respect to backup mechanisms, where portions of fresh data are stored and utilized by backups in the event of client failure.

Although the model converges faster in asynchronous federated learning, it shows a strong bias towards clients with shorter training periods. Developing feasible bias elimination approaches would constitute significant advancements in asynchronous federated learning and would improve the applicability of this framework in bias-sensitive environments.

The underlying dataset used in this work includes various attack types. While this thesis focuses on binary classification and DoS attack detection, additional research could explore multi-label classification and sophisticated attacks. Extending the framework to support other types of attacks could yield significant results and insights, advancing anomaly detection toward a wider range of attack types in real-world applications.

Overview of Generative AI Tools Used

ChatGPT (Model 4o)[1] was used for some parts of this thesis for the purposes of correcting grammar and providing suggestions for improved phrasing of content that has already been written.

[1] https://chatgpt.com/

List of Figures

List of Tables

Acronyms

AFL Asynchronous Federated Learning. 9

DoS Denial of Service. 1–4

FedAvg Federated Averaging. 1, 9

HFL Horizontal Federated Learning. 10

HIDS Host-based Intrusion Detection Systems. 7

IDS Intrusion Detection System. 1, 4

IDSs Intrusion Detection Systems. 1, 7

IIoT Industrial Internet of Things. 1–4, 6

MitM Man-in-the-Middle. 6

NIDS Network-based Intrusion Detection Systems. 7

VFL Vertical Federated Learning. 10

Bibliography

[AHS22] Haftay Gebreslasie Abreha, Mohammad Hayajneh, and Mohamed Adel Serhani. Federated learning in edge computing: a systematic survey. *Sensors*, 22(2):450, 2022.

[APMS22] Ons Aouedi, Kandaraj Piamrat, Guillaume Muller, and Kamal Singh. Federated semisupervised learning for attack detection in industrial Internet of Things. *IEEE Transactions on Industrial Informatics*, 19(1):286–295, 2022.

[B+19] Daniel Berrar et al. Performance measures for binary classification., 2019.

[BMD23] Veronika Bekbulatova, Andrea Morichetta, and Schahram Dustdar. FL-SERENADE: Federated Learning for SEmi-supeRvisEd Network Anomaly DEtection. A Case Study. In *2023 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 1072–1079. IEEE, 2023.

[BTM+20] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, et al. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[DCS24] Roberto Doriguzzi-Corin and Domenico Siracusa. Flad: adaptive federated learning for ddos attack detection. *Computers & Security*, 137:103597, 2024.

[Gaj24] Rastko Gajanin. *Navigating the Transition from Centralized to Federated Learning with Non-IID Data: A Human Activity Recognition Case Study.* PhD thesis, Technische Universität Wien, 2024.

[KGG14] Jens Keilwagen, Ivo Grosse, and Jan Grau. Area under precision-recall curves for weighted and unweighted data. *PloS one*, 9(3):e92209, 2014.

[KGVK19] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2(1):1–22, 2019.

[KMA+21] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and trends® in machine learning*, 14(1–2):1–210, 2021.

[KYF+20] Ivan Kholod, Evgeny Yanaki, Dmitry Fomichev, Evgeniy Shalugin, Evgenia Novikova, Evgeny Filippov, and Mats Nordlund. Open-source federated learning frameworks for iot: A comparative review and analysis. *Sensors*, 21(1):167, 2020.

[LAC14] Robert M Lee, Michael J Assante, and Tim Conway. German steel mill cyber attack. *Industrial Control Systems*, 30(62):1–15, 2014.

[LHY+19] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189*, 2019.

[LHZ+22] Ji Liu, Jizhou Huang, Yang Zhou, Xuhong Li, Shilei Ji, Haoyi Xiong, and Dejing Dou. From distributed machine learning to federated learning: A survey. *Knowledge and Information Systems*, 64(4):885–917, 2022.

[LLZ+22] Sin Kit Lo, Qinghua Lu, Liming Zhu, Hye-Young Paik, Xiwei Xu, and Chen Wang. Architectural patterns for the design of federated learning systems. *Journal of Systems and Software*, 191:111357, 2022.

[LSZ+20] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450, 2020.

[LT01] Neil Long and Rob Thomas. Trends in denial of service attack technology. *CERT Coordination Center*, 648(651):569, 2001.

[M. 21] M. Zolanvari, M. A. Teixeira, L. Gupta, K. M. Khan, and R. Jain. WUSTL-IIOT-2021 Dataset for IIoT Cybersecurity Research, St. Louis, USA, October 2021.

[MAF+18] Artur Marzano, David Alexander, Osvaldo Fonseca, Elverton Fazzion, Cristine Hoepers, Klaus Steding-Jessen, Marcelo HPC Chaves, Ítalo Cunha, Dorgival Guedes, and Wagner Meira. The evolution of bashlite and mirai iot botnets. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 00813–00818. IEEE, 2018.

[MMR+17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[MR04] Jelena Mirkovic and Peter Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[MYAZ15] Rwan Mahmoud, Tasneem Yousuf, Fadi Aloul, and Imran Zualkernan. Internet of things (iot) security: Current status, challenges and prospective measures. In *2015 10th international conference for internet technology and secured transactions (ICITST)*, pages 336–341. IEEE, 2015.

[Rou21] Yaman Roumani. Patching zero-day vulnerabilities: an empirical analysis. *Journal of Cybersecurity*, 7(1):tyab023, 2021.

[RVG+19] Lukas Ruff, Robert A Vandermeulen, Nico Görnitz, Alexander Binder, Emmanuel Müller, Klaus-Robert Müller, and Marius Kloft. Deep semi-supervised anomaly detection. *arXiv preprint arXiv:1906.02694*, 2019.

[SHH+20] Martin Serror, Sacha Hack, Martin Henze, Marko Schuba, and Klaus Wehrle. Challenges and opportunities in securing the industrial internet of things. *IEEE Transactions on Industrial Informatics*, 17(5):2985–2996, 2020.

[SSH+18] Emiliano Sisinni, Abusayeed Saifullah, Song Han, Ulf Jennehag, and Mikael Gidlund. Industrial internet of things: Challenges, opportunities, and directions. *IEEE transactions on industrial informatics*, 14(11):4724–4734, 2018.

[STT23] Lavanya Shanmugam, Ravish Tillu, and Manish Tomar. Federated learning architecture: Design, implementation, and challenges in distributed ai systems. *Journal of Knowledge Learning and Science Technology ISSN: 2959-6386 (online)*, 2(2):371–384, 2023.

[SWW15] Ahmad-Reza Sadeghi, Christian Wachsmann, and Michael Waidner. Security and privacy challenges in industrial internet of things. In *Proceedings of the 52nd annual design automation conference*, pages 1–6, 2015.

[TÇES25] Nasim Tavakkoli, Orçun Çetin, Emre Ekmekcioglu, and Erkay Savaş. From frontlines to online: examining target preferences in the russia–ukraine conflict. *International Journal of Information Security*, 24(1):1–15, 2025.

[WHA+16] Jacob Wurm, Khoa Hoang, Orlando Arias, Ahmad-Reza Sadeghi, and Yier Jin. Security analysis on consumer and industrial iot devices. In *2016 21st Asia and South Pacific design automation conference (ASP-DAC)*, pages 519–524. IEEE, 2016.

[WOGS17] David E Whitehead, Kevin Owens, Dennis Gammel, and Jess Smith. Ukraine cyber-induced power outage: Analysis and practical mitigation strategies. In *2017 70th Annual conference for protective relay engineers (CPRE)*, pages 1–8. IEEE, 2017.

[XKG19] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*, 2019.

[XQXG23] Chenhao Xu, Youyang Qu, Yong Xiang, and Longxiang Gao. Asynchronous federated learning on heterogeneous devices: A survey. *Computer Science Review*, 50:100595, 2023.

[YWY+17] Yuchen Yang, Longfei Wu, Guisheng Yin, Lijie Li, and Hongbin Zhao. A survey on security and privacy issues in internet-of-things. *IEEE Internet of things Journal*, 4(5):1250–1258, 2017.

[ZBO20] Hongyi Zhang, Jan Bosch, and Helena Holmström Olsson. Federated learning systems: Architecture alternatives. In *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*, pages 385–394. IEEE, 2020.