DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Robotics, Cognition, Intelligence

Blockchain-Based Voting for Updating Predictive Maintenance Models in Federated Learning

Blockchain-basierte Abstimmung zur Optimierung von vorausschauenden Wartungsmodellen im Rahmen des föderierten Lernens

Author:	Carlos Xavier García Briones
Supervisor:	Prof. Dr. rer. nat. Ruben Mayer
Advisor:	M.Sc. Pezhman Nasirifard
Submission Date:	15.03.2022

I confirm that this master's thesis in robotics, cognition, intelligence is my own work and I have documented all sources and material used.

Munich, 15.03.2022 Carlos Xavier García Briones

Acknowledgments

I would like to express my very great appreciation to MSc. Marisa Mohr and Dr. Robert Pesch, my external supervisors, for their valuable and constructive suggestions during the planning and development of this thesis.

I would also like to thank inovex GmbH for providing me the support and technical resources to perform the experiments of this work.

My grateful thanks are also extended to MSc. Pezhman Nasirifard, my supervisor, for his patient guidance and useful critiques of this work.

I am particularly grateful for the assistance given by MSc. Justus Drögemüller in the review of this thesis.

Finally, I wish to thank my parents and sisters for their support and encouragement throughout my study.

Abstract

This work analyzes the integration of a permissioned blockchain network for a verification mechanism to perform decentralized, robust, and privacy-friendly federated learning (FL). In particular, a cost-effective variant of the state-of-the-art (SOTA) approach is proposed to address vulnerabilities of federated learning, such as a compromised server, compromised clients, non-robust aggregation, and gradient leakage. The open-source environment Hyperledger Fabric is used for implementation.

In our approach, we analyze the resource overhead of Hyperledger Fabric components and derive the blockchain costs. Assuming a monthly FL session with ten clients, we achieve an annual blockchain cost of about 0.50% of the annual blockchain cost of the state-of-the-art approach. In addition, the performance and robustness of the approach were analyzed to predict the remaining useful life of turbofan engines. Data were distributed among ten participants, and a long-short-term memory model was trained. The mean square error (RMSE) was considered as a quality measure. Differential privacy is used as a method to ensure privacy. In this regard, this paper analyzes the effect of specific privacy budgets for different models. The privacy budget's effect on the central model's performance becomes more negligible for a higher number of participants. We also analyze the robustness of our approach to collusion between compromised participants of the blockchain network. The tests performed prove robustness comparable to the SOTA implementation. Finally, we investigate the use of percentage deadline criteria for submitting local models. We consider the worst-case scenario in which valuable data is distributed to the last percentage of clients who have completed their training. We find that a percentage deadline criterion of 80% is a good compromise between the robustness of the approach and the performance achieved.

Zusammenfassung

Diese Arbeit analysiert die Integration eines permissioned Blockchain-Netzwerks für einen Prüfmechanismus zur Durchführung von dezentralem, robustem und datenschutzfreundlichem föderiertem Lernen (FL). Insbesondere wird eine kosteneffiziente Variante des State-of-the-Art (SOTA) Ansatzes vorgeschlagen, um Schwachstellen des föderierten Lernens, wie beispielsweise einen kompromittierten Server, kompromittierte Clients, nicht-robuste Aggregation und Gradient Leakage zu beheben. Zur Umsetzung wird die quelloffene Umgebung Hyperledger Fabric verwendet.

In unserem Ansatz analysieren wir den Resourcenaufwand von Hyperledger Fabric-Komponenten und leiten daraus Blockchain-Kosten ab. Unter der Annahme einer monatlichen FL-Sitzung mit zehn Teilnehmern erreichen wir jährliche Blockchain-Kosten von etwa 0,50% der jährlichen Blockchain-Kosten des SOTA Ansatzes. Darüber hinaus wurden die Leistung und Robustheit des Ansatzes analysiert, um die verbleibende Nutzungsdauer von Turbofan-Triebwerken vorherzusagen. Die Daten wurden auf zehn Teilnehmer verteilt und ein Long-Short-Term-Memory-Modell wurde trainiert. Als Qualitätsmerkmal wurde der mittleren quadratischen Fehler (RMSE) betrachtet. Differential-Privacy wird als Methode zur Sicherstellung des Datenschutzes verwendet. Diesbezüglich wird in dieser Arbeit die Auswirkung bestimmter Datenschutzbudgets für unterschiedliche Modelle analysiert. Für das zentrale Modell gilt in der Regel je höher die Anzahl der Teilnehmer, desto geringer sind die Auswirkungen des Datenschutzbudgets auf dessen Leistung. Außerdem analysieren wir die Robustheit unseres Ansatzes gegenüber geheimer Absprache zwischen kompromittierten Teilnehmern des Blockchain-Netzwerkes. Die durchgeführten Tests beweisen eine Robustheit vergleichbar zu der SOTA Implementierung. Schließlich untersuchen wir die Verwendung von prozentualen Fristenkriterien für die Einreichung lokaler Modelle. Wir betrachten das Worst-Case-Szenario, in dem wertvolle Daten an den letzten Prozentsatz der Klienten verteilt werden, die ihr Training abgeschlossen haben. Wir stellen fest, dass ein prozentuales Fristenkriterium von 80% einen guten Kompromiss zwischen der Robustheit des Ansatzes und der erreichten Leistung darstellt.

Contents

Acknowledgments **iii**

Abstract **iv**

Zusammenfassung **v**

1. Introduction **1**
1.1. Problem Statement 2
1.2. Approach 3
1.3. Contributions 3
1.4. Organization 4

2. Background **5**
2.1. Predictive Maintenance 5
2.2. Blockchain Technology 6
2.2.1. Permissioned Vs. Permissionless Blockchain 7
2.2.2. Blockchain Platforms 7
2.2.3. Hyperledger Fabric 8
2.3. Federated Learning 14
2.3.1. Vulnerabilities 15
2.3.2. Attacks 17
2.3.3. Defenses 17

3. Related Work **20**
3.1. Decentralization 20
3.2. Privacy 21
3.3. Quality Control 21
3.4. Incentive Mechanism 22
3.5. Accountability and Authentification 22
3.6. State of the Art 23

4. Hyperledger Fabric Network **24**
4.1. Fabric Network Deployment 25

4.2. Fabric Network Configuration 26

5. HyperFlow **29**
5.1. HyperFlow Client 29
5.1.1. Cloud Client 30
5.1.2. Crypter 31
5.1.3. Deep Learning Module 31
5.1.4. Data Loader 31
5.1.5. Blockchain Connector 31
5.1.6. Deadline Manager 32
5.2. Federated Learning Workflow 32
5.2.1. Phase 1: Client Ready 33
5.2.2. Phase 2: Training 34
5.2.3. Phase 3: Validation 35
5.2.4. Phase 4: Evaluation 35
5.2.5. Phase 5: Score Decryption 36
5.2.6. Phase 6: Aggregation 37

6. Experimental Setting **39**
6.1. Data Set 39
6.2. Predictive Maintenance Model 40

7. Results and Discussion **43**
7.1. Hyperledger Fabric Network 43
7.1.1. Network Benchmark 43
7.1.2. Resource consumption 45
7.1.3. Costs 47
7.2. HyperFlow 48
7.3. Differential Privacy 51
7.4. Malicious Clients Threat Analysis 53
7.5. Percentage Deadline Criteria 55

8. Conclusion **58**

A. Configuration File **61**

List of Figures **63**

List of Tables **65**

Bibliography **66**

1. Introduction

Data-driven products with the aim of predictive maintenance have become increasingly important in recent years. They can optimize the lifespan of a machine, reduce maintenance costs or minimize unplanned downtime. Thus great potential for cost savings exists. [1]

Research with the focus on predictive maintenance is being conducted primarily in the area of smart factories. However, new business models, such as machine leasing, can also profit from it [2]. Furthermore, with growing capabilities of data ingestion, increasing computing power, and the possibility of cloud computing, machine learning models have increased their performance in real-world environments [3][4].

For machine learning models to be implemented in real-world predictive maintenance problems and achieve outstanding performance, it is necessary to gather high-quality data sets [4]. Gathering such data sets is commonly challenging and costly. A single user, which can be, for example, an institution, a firm, a machine, or a person, has a limited amount of high-quality data. A viable approach to collect such data sets is to unite data from several users. This is difficult in most cases due to data being sensitive and private. Several techniques have been explored for privacy-preserving data mining, such as data anonymization, data perturbation, data randomization, data condensation, and cryptography. Each of them contains its limitations [5]. For example, data anonymization, i.e., removing personal information from data, would be considered trivial at first glance. However, it has been proven that in some cases, the use of publicly available datasets can be used to deanonymize data or at least reveal certain privacy aspects [6][7]. Altogether, good performing ML models for predictive maintenance can be trained using the collective data gathered by many users. When designing such an approach, the privacy and sensitivity attributes of data have to be considered.

Federated Learning (FL) addresses privacy concerns when sharing data by bypassing the retrieval and centralization of users' data. The concept of FL is to use a central server (curator) that distributes a copy of the central model to the participating instances (clients), receives model updates from the clients' local model, and aggregates them in a valuable way to increase the overall performance of the central model. Clients train the local models with their data and infrastructure, and later send the updates to the

curator. Data security is assured by sharing only model updates and using differential privacy [8]. In addition, secure aggregation protocols such as the secure multi-party computation are used to ensure the clients' anonymity [9].

Although FL provides a solution for creating valuable ML models while using decentralized data, it has not been as widely integrated into the ML community [10]. One reason for this is the vulnerability of this approach to security and privacy attacks. Some of these attacks, such as gradient leakage and data reconstruction through inference, have already been addressed, e.g., by using differential privacy.

When implementing FL into a predictive maintenance environment, crucial concerns not addressed until now are the accountability and resiliency of the approach to (i) malicious clients and (ii) a malicious curator [11]. In the first case, clients would act according to their own hidden agenda, which means they have their interests in mind and not the one of the consortium. For example, in a consortium where a classification of broken machine parts is the shared task, a client could easily interchange the labels in their dataset, train a model that classifies broken parts as fully functional and send that model to the curator to be aggregated with all other clients' model. This would negatively impact the central model, and there would be no accountability for which client provides a valuable model. In the second case, the curator provides a single point of failure for the FL framework. For example, the curator could be compromised and modify the FL protocol to obtain private information about individual clients.

Mugunthan et al. [12] developed a protocol for an evaluation round between clients, which provides accountability for the FL process and create a decentralized curator through the use of the Ethereum blockchain. They named their approach BlockFlow. Their main contribution is the evaluation protocol between clients. Each client evaluates the other clients' model, generates a score for each model with all clients' evaluation results, and aggregates its own central model. This makes it possible to check whether the central model matches the central model computed by other clients.

1.1. Problem Statement

Nevertheless, the proposed approach of Mugunthan et al. [12] has several shortcomings. Firstly, the deadlines controlling each stage in the experiment are hardcoded. For each FL round, the time for the training phase has to be set large enough for local models to converge. The training time for each client depends on several factors, such as the available hardware, task, model size, dataset's size, and dataset's quality, which translates to the epochs needed to find a suboptimal solution. The time parameter for

the FL round has to be empirically adjusted for each network topology. Secondly, with the current cost of an Ethereum's native coin of 2,935.36 Euro[1], an FL session containing ten rounds as proposed by Mugunthan et al. with ten clients has a blockchain cost of approximately 1,568.79 Euro. Additionally, the costs are proportional to the number of clients present and the rounds done in a session. Furthermore, it does not include the computation cost needed to train the ML model.

1.2. Approach

In this work, we implement a cost-efficient variant of the state-of-the-art BlockFlow scoring procedure to generate an FL framework in a predictive maintenance environment. The blockchain costs are held to a minimum by interchanging the public Ethereum blockchain framework by the permissioned blockchain framework Hyperledger Fabric. The throughput and failure of transactions are evaluated according to an increasing number of clients. Additionally, the costs of running an FL session are examined. To maintain results independent of the used ML model, the training of the ML model is done in a separate Docker container.

It has been shown that the training phase in FL can be drastically slowed down by a few machines taking a long time [13]. Setting a hard time constrain to this phase has to be optimized for each task, model, available hardware, and data available to the clients. A percentage rule [14] is used to continue to the validation phase. This approach is evaluated, and the selection of optimal deadline criterion for FL rounds is reviewed in the predictive maintenance environment [15].

1.3. Contributions

The contributions of this work can be summarized as follows:

- Architecture design, implementation, and evaluation of a cost-efficient variant of the BlockFlow approach in Hyperledger Fabric
- Evaluation of the FL permissioned blockchain approach in the context of predictive maintenance using a public data set called Turbofan Engine Degradation Simulation Data Set provided by NASA [16]
- Evaluation of the percentage rule as deadline criterion for the submission of local models in FL

[1] https://www.coinbase.com/de/price/ethereum. Visited on: 17.01.2022

1.4. Organization

Chapter 2 provides an introduction to predictive maintenance, the used blockchain platform, and the vulnerabilities and attacks present in federated learning. Furthermore, work related to the proposed approach is presented in Chapter 3. The framework developed for the automatic deployment of the Hyperledger Fabric network is detailed in Chapter 4. Moreover, the proposed approach, together with its workflow and components, is shown in Chapter 5. The experimental setting, which includes the data set utilized and predictive maintenance model used, are presented in Chapter 6. After that, Chapter 7 presents the evaluation of the proposed approach and analysis of the contributions of this thesis. Finally, Chapter 8 summarizes the results and insights gained by this work.

2. Background

This chapter reviews the background material related to this thesis. First, the field of predictive maintenance is presented, which we use as a use case for this work. Thereafter, relevant technologies such as blockchain technology and federated learning are introduced. As blockchain is a broad topic, the review is limited to the Hyperledger Fabric platform [17], which is used in this work. Finally, a brief introduction to federated learning is given. It includes an overview of the vulnerabilities, attacks, and defenses in the context of federated learning.

2.1. Predictive Maintenance

With the increased information gain by the adaption of the Industrial Internet of Things in machines, new opportunities for cost-efficient operations emerge [18]. One-third of the maintenance costs are wasted due to unnecessary or improperly carried out maintenance. Typical maintenance management is divided into two categories: (i) reactive maintenance and (ii) preventive maintenance. On the one hand, reactive maintenance management is characterized by high spare parts inventory cost, high machine downtime, and high overtime labor costs. It is the most expensive maintenance management in the long run. Overall, it includes a higher cost for either inventory or the fast delivery of spare parts. It also has the highest amount of machinery downtime. On the other hand, the management programs used for preventive maintenance are time or usage driven. There is no closed loop for the information flow from the machine to the maintenance schedule. Maintenance is scheduled using, e.g., a statistical approach to the time-to-failure relation of a machine. One of the downsides of this type of maintenance management is the scheduling of unnecessary preventive measures. Another more costly downside is the possibility of the machine failing before scheduled maintenance. This implies that the fixes will be made following a reactive maintenance model while maintaining the high costs of unnecessary scheduled measures. [19]

Predictive maintenance is the management of a condition-driven preventive maintenance scheduling using a closed information loop. This refers to the use of non-destructive techniques for monitoring the mechanical conditions, performance, and other indicators of a system to reduce the scheduling of maintenance procedures while

having a low machinery downtime [19]. A challenge of predictive maintenance is the generation of a system's health indicators and finding the relationship between their operating costs and failure risk [20]. Machine learning algorithms have proven to be able to provide state-of-the-art solutions in this area [21, 22].

In this thesis, we evaluate the proposed FL approach in predictive maintenance. A Long-Short Term Memory (LSTM) model is used to predict the remaining useful lifetime of Turbofan motors. An LSTM is a type of recurrent neural network, which has the capability of retaining information to use it for predictions in the short and long term [23].

2.2. Blockchain Technology

Blockchain technology is widely known for its use in the world of cryptocurrencies. It was initially implemented as a solution to the dependence on a centralized financial authority, which gave birth to Bitcoin [24] in 2008. Conversely, other blockchain platforms focus on the use of smart contracts. Smart contracts are legal agreements previously consented by the participating parties, which are autonomously enforced through the definition of the agreements as executable logic [25]. Using blockchains to implement smart contracts provides trustworthiness to the transactions performed [26].

Blockchain technology is based on a forgery-proof and tamper-proof decentralized database called a ledger. It is an append-only database that keeps track of all transactions done within the network, and each user in a blockchain network has an identical copy of it. The method used to keep the ledgers synchronized is called the consensus mechanism. The consensus mechanism varies from one blockchain platform to another. Under usual conditions, scheduled transactions should be written in the same order to every ledger. In practice, ledgers can write transactions in a different order and thus fork from the original blockchain network. The used consensus mechanisms try to avoid such problems. [27]

To perform transactions, the blockchain uses asymmetric-key cryptography. Transactions must be first signed and then scheduled to be committed to the blockchain. This type of cryptography enables the establishment of trust between pseudo-anonymous users. In addition, it provides a mechanism for transactions to be public and at the same time to verify their authenticity and integrity. Private keys are used to digitally sign transactions, while public keys are used to verify the integrity and authenticity of the signature. [28]

The most common consensus mechanisms for public networks are Proof of Work (PoW) and Proof of Stake (PoS). PoW relies on solving a cryptographical puzzle. The puzzle is the following: find a nonce, i.e., a natural number, that by adding it to the block's header, the resulting hash has a predefined number of leading zeros. Finding the solution is computationally expensive, and the use of more hardware resources increases the possibility of finding the nonce faster but does not guarantee it will be found before someone else solves the puzzle. On the other hand, PoS selects the endorsing node with regard to a sample over a weighted probability distribution. The weights are given by the amount of cryptocurrency a node has. This amount of cryptocurrency given as the Stake will need a cool-down time in order to be used again. [29]

2.2.1. Permissioned Vs. Permissionless Blockchain

Depending on the rights needed for accessing a blockchain network, it can be distinguished between a permissionless or a permissioned network. A permissionless network has several downsides to it. Since they allow any entity to be part of the network, the algorithms used for determining who validates a transaction are very energy and time inefficient. This limits the speed of processing large volumes of transactions. Conserving privacy is also one of the issues in these networks. The network guarantees pseudo-anonymity by assigning each user a key. Since all transactions done over the network are published in every ledger, knowing the key of an entity would result in also knowing each transaction this entity has made since it joined the network. [30]

Permissioned networks have restrictions in their memberships and procedures. This lowers the privacy concerns and provides, in general, a higher transaction throughput to the network. Since the base trust in such networks is higher than in a permissionless one, it allows the usage of more efficient algorithms for transaction validations. Each member has a role and authorizations assigned to it. This leaves a possibility for undermining the credibility of the blockchain if the network is set up incorrectly, e.g., an entity could be given override privileges. [30]

2.2.2. Blockchain Platforms

Since the interest of this thesis is set on smart contracts blockchain platforms to provide methods for generating transaction proposals in the network, this section will briefly present some of them. On the one hand, permissionless smart contract platforms with

native cryptocurrencies include Ethereum[1], Binance[2], Solana[3], Cardano[4], Polkadot[5], among others. On the other hand, popular permissioned blockchain platforms are Hyperledger Fabric[6], Ethereum Enterprise[7], Quorum[8], MultiChain[9], and R3 Corda[10]. Polge et al. [31] compared the mentioned platforms taking into consideration the blockchain's privacy, scalability, transaction throughput, adoption and latency. The adoption attribute reflects the industrial use cases implemented and the amount of scientific interest in the corresponding platform. The privacy attribute concerns itself with the data exposure to the participants and the transaction privacy. In a permissioned blockchain, participants are known. Therefore, the transactions made and data shared should have restrictive access to the participants in the network. Overall, it is seen that Hyperledger Fabric has the highest score at privacy, throughput, and latency while being the second-best in the other attributes. On the other hand, the permissioned Ethereum blockchain achieves a better scoring in the scalability and adoption attributes while performing poorly in the latency and privacy attributes. All other mentioned platforms have lower performance in all attributes than the Hyperledger Fabric platform. [31]

2.2.3. Hyperledger Fabric

Hyperledger Fabric [17], from now on referred to as Fabric, is a framework for creating permissioned blockchain networks. It is open source and hosted by the Linux Foundation. Production environments are increasingly using Fabric for blockchain use cases in industrial fields such as the Internet of Things [32, 33, 34]. A distinctive feature of Fabric is the actual knowledge of entities doing each transaction. Entities are grouped into organizations, which are assumed to be untrusty of each other. An organization can be divided into organizational units (OU), also called affiliations. This subdivision is helpful for giving different authorization rights to each of these units of an organization. Every entity in an organization fully trusts all other entities and OUs in the same organization.

[1] https://ethereum.org/
[2] https://www.binance.com/
[3] https://solana.com/
[4] https://cardano.org/
[5] https://polkadot.network/
[6] https://www.hyperledger.org/use/fabric
[7] https://ethereum.org/en/enterprise/
[8] https://consensys.net/quorum/
[9] https://www.multichain.com/
[10] https://www.r3.com/corda-platform/

Channel

Channels are elements of the Fabric network that allow a set of the consortium's organizations to communicate privately. They provide means for sharing infrastructure and data while keeping it private from other channels and the network. Each channel can have its own policies definition and restrict access to a set of network members.

An essential element of the sharing infrastructure that channels allow is the sharing of smart contracts, which are the same as chaincodes in Fabric. A smart contract defines an executable logic used to interact and update a channel's ledger. Smart contracts are written in the programming languages Javascript, Typescript, Go, and Java. They are primarily used to perform PUT, GET and DELETE requests to the ledger's key-value database, known as the world-state, and they can also be used to query the blockchain's record of transactions. Nodes connected to a channel will be able to install the chaincodes committed to the channel and have a copy of the channel's ledger. Fabric versions 2.x have the characteristic that if a chaincode is not approved by an organization and still committed, it will not be installed in the peers from organizations that rejected the chaincode. [35]

Peers

One of the elementary nodes provided by Fabric is the peer. It can join channels and interact with its copy of the channel's ledger through smart contracts committed to the channel and accepted by the peer's organization. Applications use methods implemented on installed chaincodes to generate transactions to modify or read the channel's ledger.

The distributed ledger consists of the world-state and the chain of blocks. The motivation behind this structure is having the present state of a set of key-values in a ledger and preserving the actions taken since the beginning to reach these values. The world-state is a set of versioned key-value pairs stored as a database, which is currently (Hyperledger Fabric v2.4) implemented in LevelDB[11] or CouchDB[12]. The block-chain itself consists of blocks storing transactions in them. [35]

As shown in Figure 2.1, each block contains a header, the block data, and the block metadata. The header contains a block number, which is implemented as an ascending counter, a cryptographic hash from the block data, and the cryptographic hash of the previous block. This characteristic makes the modification of previous blocks in the block-chain detectable. It creates a Merkle tree with the header hashes of the blocks and the block's data. Changing a previous block will provoke a different header of the

[11]https://github.com/google/leveldb
[12]https://couchdb.apache.org/

most recent block. The network can automatically address this inconsistency, and the ledger would be synchronized to the consented channel's ledger. The initial block is called genesis. [35, 36]

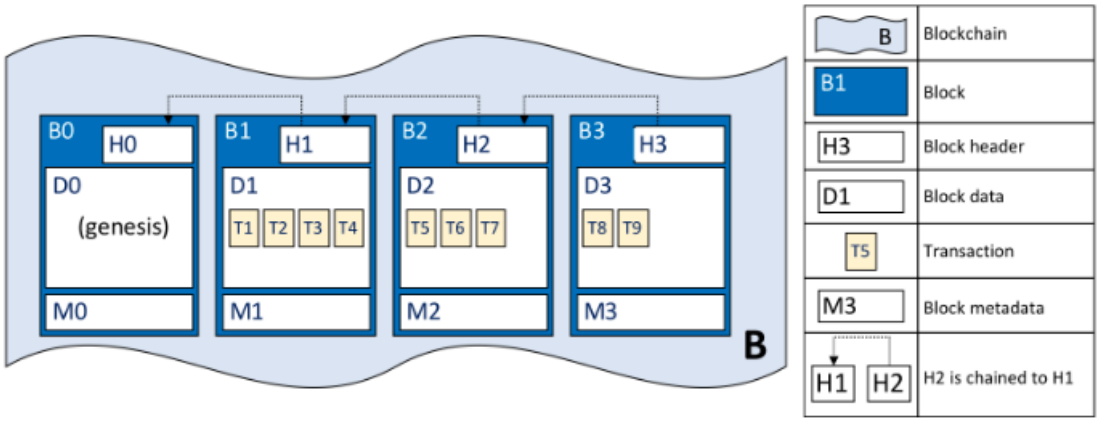

Figure 2.1.: Description of the block-chain in Hyperledger Fabric. Source: [35]

The block's data contains a list of ordered transactions. As presented in Figure 2.2, a transaction consists of a header, which contains metadata of the creation of the transaction, a cryptographic signature created using the client's private key, a proposal containing the parameters passed from the application to the smart contract, a response as a Read-Write set, which captures the changes made in the current world-state and the values after the changes are applied, and the endorsements, which is a list of signed transactions by the endorsers necessary to fulfill the consensus policy. [32]

Ordering Service

The ordering service is responsible for packing transactions in a defined order in a block and distributing the block to the available peers. This service is composed of ordering nodes called orderers. It also contains a list of the organizations allowed to create a channel in the network.

There are three implementations of the ordering service in Fabric v2.x: Raft, Kafka, and Solo. The two latter ones are deprecated but still supported. The Raft and Kafka implementation are crash fault-tolerant (CFT) and follow a "leader and follower" model. Although Kafka can have several instances of ordering nodes to be CFT, they must be managed by the same organization. This attribute reduces the decentralization property of the network. On the other hand, Raft allows different organizations to participate in the ordering service by creating their instance of an ordering node. The leader, in

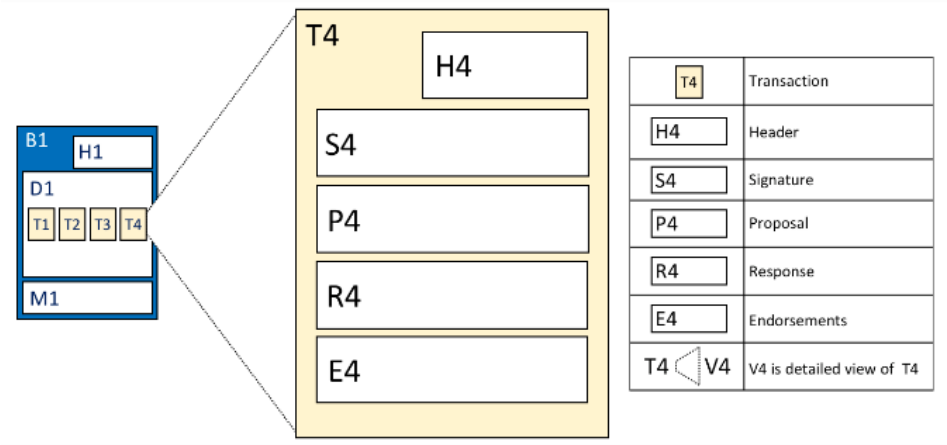

Figure 2.2.: Illustration of a transaction's components in Hyperledger Fabric. Source: [35]

this case, is dynamically selected among the orderers in a channel. The leader then replicates the messages received to the follower nodes. The CFT ability allows the correct performance of the ordering protocol while the majority of the ordering nodes are available. [35]

Policies are how Fabric decides what decision is made and who is authorized to do specific actions. There are three levels in the policy hierarchy. System channel policies define the consensus needed for an organization to join the network and for a change in the blockchain structure to be accepted. Application channel policies define the consensus needed to be reached for transactions in networks to be accepted, new members to be accepted in the channel, and policies for the organizations' nodes. Access control lists and chaincode policies define mechanisms to restrict access to data and chaincodes on a channel. [35]

These policies go hand in hand with the configurations done at each hierarchy level. The system channel configuration contains the organizations that are part of the ordering service (ordering organizations) and those that are allowed to perform transactions (consortium organizations). It also defines the consensus used by the ordering service and how new blocks are created. The application channel configuration governs the life-cycle of a chaincode in a channel. It defines which organizations are required to approve a chaincode implementation in order for it to be committed in a channel. Access control lists can configure the access from resources to specific methods defined on a chaincode. [35]

Reaching consensus over a network is defined by three main aspects: the endorsement, ordering, and validation of transactions. Figure 2.3 shows the flow of how transactions are committed in Fabric. First, an application connects to the peer and invokes a method from the chaincode. It sends a transaction proposal to the endorsing peers needed to fulfill the consensus policy. The peers will compute a response proposal containing the operations needed to update the ledger. This response is signed and returned to the application. Later, the transaction proposal and the endorsed responses are sent to the ordering service. The ordering service will pack several transactions in a block and distribute them to the available peers. The amount of transactions in a block depends on the parameters defined in the channel configuration. These parameters are either the maximum size in Bytes that a block can have or the maximum time an ordering service will wait to create a block. Fabric differs from other blockchain platforms in that the order of blocks stated by the ordering service will be final. In that way, there will be no differences between peers' ledgers, also called ledger forks. If a peer is not available at the time of the distribution of a new block, it can still receive the block in two ways. Either at the time of connecting again to the ordering service or through a gossip protocol between peers. After the orderer has distributed the new block, peers will validate and commit the transactions received in a block. Each transaction in a block will be validated by each peer independently. In this validation process, peers verify that transactions meet the endorsement policy defined in the channel and a transaction has not been invalidated by another transaction that could have been in-flight when the transaction was endorsed. Failed transactions will be kept in the block but marked as invalid. Invalid transactions will not update the ledger's state. The validation and committing process is a deterministic process and ensures that all peers have an identical ledger. In the last step, the application will receive a notice that the ledger has been updated. [35]

Certificate Authorities

Fabric follows a structure managed by organizations. Each component must be owned by an organization, and its ownership must be verifiable. For this, each organization has its certificate authority (CA). The responsibility of the CA is to generate the local Membership Service Provider (MSP) for each node, which can be a peer or an orderer. Nodes' local MSPs are then used to create a channel's MSP, which other members of the channel use to verify the identity of each node. OUs are composed of members such as peers, orderers, and users. They have to register through the corresponding CA and enroll in the corresponding MSP. In the registration of the component, the affiliations and roles of the member are set. Thereafter, generating an identity is done by generating a private and public key. The local MSP of the member contains the CA

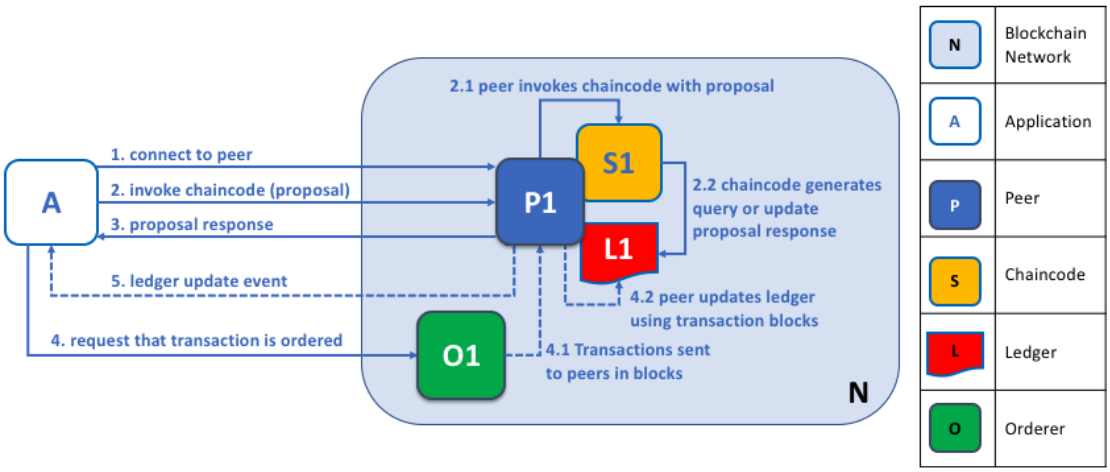

Figure 2.3.: Workflow of a submitted transaction in Fabric. Source: [35]

certificate, TLS-network certificates, intermediate certificates, enrolled users, the public key of the component, and the private key. [35]

Network

Figure 2.4 shows an example of a blockchain network in Fabric containing four organizations. Organizations R1, R2, and R3 have a peer node, and organization R4 an ordering node. Each organization has a certificate authority that provides certificates for the corresponding nodes. Two channels allow organizations that joined the channel to have a distributed synchronized copy of the channel's ledger. In the same way, each channel has a chaincode that provides methods for applications to interact with the channel's ledger through their peer nodes. For a chaincode to be committed into a channel, the organizations in the channel have to approve it. It can therefore be seen that organizations in the channel have control over the committed channel's chaincode. In the same way, for the instantiating of the network, a network configuration has to be given to the ordering nodes. This shows that at the beginning, the network was created by organizations R1 and R4. Other organizations can later join the network but need to meet the predefined consensus in the network's configuration.

The network configuration is the first step for generating a Fabric network. It specifies the organizations taking part and the policies for reaching consensus on different levels, such as transaction endorsement, channel-wise, network-wise, and several types of actions. It also provides the initial configuration of the network, such as the ordering service type, which nodes belong to the network, and the overall MSP of the network.

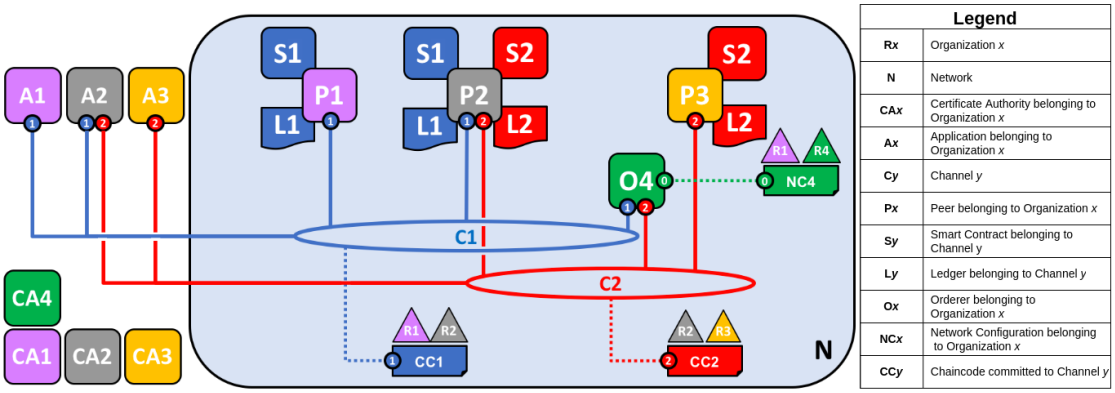

Figure 2.4.: Example of a Fabric network containing four organizations and two channels. Figure was adapted from [35]

2.3. Federated Learning

The term Federated Learning (FL) was introduced in 2017 by B. McMahan et al. [14]. The methodology was developed for the learning of a model's parameters with decentralized training data in mobile devices.

Traditional machine learning techniques, e.g., models to predict the remaining useful lifetime in predictive maintenance, use centralized data. More specifically, traditional machine learning requires collecting and storing information in a local or cloud data center. This not only implies costs for the harvesting of data but also for its storage. More significant are the risks of storing private data in a centralized location [11]. FL addresses this issue by enabling devices to collaborate in the optimization of a shared central model by sharing the parameters of a locally trained model with locally available data. This is done in an iterative process orchestrated by a central server, called the curator. Participants in the FL process are called clients.

As mentioned, FL is an iterative algorithm consisting of rounds to optimize a shared central model. Each training round consists of the following steps: (i) Client Selection, (ii) Central model broadcast, (iii) Local training, (iv) Aggregation, and (v) Updating the global model. In the client selection phase, a set of clients is sampled from available clients meeting the eligibility criteria defined. In the second step, the curator sends the central model to the selected clients. Thereafter, each selected client trains the received model with their local data. In the aggregation phase, clients send their local model's parameters to the curator, which are aggregated according to the predefined

aggregation algorithm. The last step is the updating of the central model by the curator.

The FL approach can be distinguished into three settings. One of them is the data-center distributed learning, where data is centrally stored and can be shuffled and distributed in a balanced manner across clients. In this case, data privacy is not an issue, and the distributed optimization property of FL is the most valuable asset. Another setting is the Cross-silo FL. In this case, clients generate their data locally, which remains decentralized. No silo has access to data generated by another silo. Also, the client availability is high, and their used hardware is homogeneous. The third setting is the cross-device FL. As in the cross-silo setting, data is generated by the client and unseen by other clients, but the used hardware is heterogeneous. [11]

While data security is locally assured, FL introduces new surfaces of attack [37]. For example, the sharing of trained parameters leaves a spot open to attacks at training time. Another source of the vulnerability of this approach is the possibility of a compromised client or a compromised curator. On the privacy side, the communication channels between clients and curator are also of concern. For example, if a client's model update is available to a malicious participant, inference attacks to reveal training data can be applied. Therefore, vulnerabilities in the approach have to be taken into account when designing and orchestrating FL. [38, 10]

In order to create a holistic view of the threats in FL, the approach's vulnerabilities, known attacks, and researched defenses are presented in the following sections.

2.3.1. Vulnerabilities

The vulnerabilities of the FL approach can be segmented into security concerns and privacy concerns [10]. Security concerns itself with the system's correct behavior, integrity, and efficiency against malicious external influences [38]. Furthermore, privacy concerns itself with the access restriction and non-disclosure of data to unauthorized influences [38]. Ideally, each component has only access to the information required to perform its operations. Sadowsky et al. [39] define vulnerability as "a flaw or weakness in a system's design, implementation, or operation that can be exploited by an intruder to violate the system's security policy" [39].

Communication Channel

Each FL session is composed of several FL rounds. On each round, communication between the clients and the server is established to broadcast the central model and the local models. The channel used for communication can be a source of vulnerability.

Eavesdroppers could intercept the models being exchanged and replace these with malicious models. Future attacks could be performed on the extracted local models and provide statistical insights into a client's data. [40]

Gradient Leakage

Although FL addresses the privacy aspect of data by sharing only locally trained model parameters, these model parameters are susceptible to gradient leakage. Performing gradient leakage attacks has been shown to reveal sensitive information about a client's local data. Furthermore, the parameter's distribution in model updates can be learned in order to create malicious models, e.g., models performing a malicious side-task. These models can be used to replace genuine models in the FL workflow and make it challenging for curators to detect them. [41]

Compromised Clients

The distributed nature of the FL approach gives clients a greater capacity to influence their data and training process. For example, a corrupted client could tamper with its training process producing malicious models or incompatible models to disrupt the improvement of the central model. Furthermore, training data could be modified to alter the central model in a targeted direction. Moreover, having access to the deployed central model presents an opportunity for malicious agents to infer features correlated to a specific model's output and craft adversarial samples. [10, 37]

Compromised Server

FL approaches involve a centralized server that presents a single point of failure for the whole approach. The server is responsible for orchestrating the training process by specifying training parameters, broadcasting the central model, and aggregating the model update of the clients. A compromised server can replace the central model with an arbitrary model and broadcast it to the clients. Access to all clients' gradient updates allows malicious agents to reveal clients' sensitive information. Moreover, the aggregation process could be altered to give a malicious client's model preference. [10, 37]

Aggregation Algorithm

The aggregation algorithm is responsible for updating the central model. Cross-device FL integrates the model's training in heterogeneous devices. The aggregation algorithm should be robust enough to provide fairness in this setting. Furthermore, clients

selected for an FL round could at some point drop out and be unresponsive. Non-robust algorithms present a vulnerable surface for malicious clients to increase their impact by undermining benign client's model update or by a benign client's dropout. [10, 42]

Distributed Nature

The distributed nature of FL presents a vulnerable surface for multiple parties to collude in a coordinated attack. Additionally, the coordination of the distributed rounds has to be carefully configured to consider sources of bias. For example, setting up an FL environment to train a natural language processing model using data in smartphones needs to take into account the device's status, such as the battery and internet connection, to participate in an FL round. For example, stating to be in a wireless connection and charging as requirements for the participation in the FL round introduces a time-zone bias. Smartphones usually meet these requirements at night, meaning that the devices participating in the FL round are from a particular time zone. The produced model will also capture this bias. [37]

2.3.2. Attacks

A malicious actor uses attacks to take advantage of one of the mentioned vulnerabilities. On the one hand, attacks focusing on retrieving sensitive information are inference attacks, Generative Adversarial Network (GAN) reconstruction attacks, and a malicious server. On the other hand, the central model can be corrupted by attacks such as poisoning attacks, malicious server, non-robust aggregation, and Man-in-the-Middle attacks. Additionally, the used FL framework has to be robust against attacks with the capability to disrupt the FL process, such as dropout of clients, communication bottlenecks, training rules manipulation, and a malicious server. Successful attacks can provide sensitive information of clients, manipulate the central model or disrupt the FL approach. Table 2.1 summarizes known attacks.

2.3.3. Defenses

Known defenses to the attacks mentioned in Table 2.1 are summarized in Table 2.2. This thesis uses differential privacy, anomaly detection, and a robust aggregation algorithm to defend against the following attacks: data poisoning, model poisoning, non-robust aggregation, inference attacks, and GAN reconstruction attacks. Differential privacy is a method for maintaining the statistical properties of data while adding statistical noise to it [8]. Anomaly detection defenses employ methods for detecting models not contributing to the increase in performance of the central model [43, 44, 45]. Malicious

Table 2.1.: Summary of federated learning attacks. Source: [37]

Attacks	Description	Sources of Vulnerabilities
Data Poisoning	manipulate client data, perhaps by flipping labels or modifying specific features of the data	• Compromised Clients
Model Poisoning	manipulate model updates to bias the global model towards a certain objective	• Compromised Clients • Compromised Server
Backdoor Attacks	insert hidden backdoors into the global model while retaining the accuracy of the main task	• Compromised Clients
Evasion Attacks	circumvent a deployed model by carefully manipulating the data samples fed into it	• Compromised Clients • Model Deployment
Non-Robust Aggregation	aggregation algorithms with weak defense mechanisms or ill-disposed reweighting schemes that cause the global model to behave abnormally	• Aggregation Algorithm
Training Rules Manipulation	manipulating model training rules such as training hyperparameters to prevent convergence or introduce bias in the trained model	• Compromised Clients • Distributed Nature
Compromised FL Distributed Computation	diverging from executing the wanted behavior of distributed FL computation to generate unreliable intermediate results	• Compromised Clients • Compromised Server • Distributed Nature • FL Environment Scope
Inference Attacks	analyze information leaked about participants in order to illegitimate gain knowledge about the FL process (e.g., participants, data, features) and use this knowledge to craft an attack against FL	• Compromised Server • Communication
GAN Reconstruction Attacks	employ GANs to recover synthetic samples of the training data through inference and use these samples to poison the training data	• Compromised Clients • Communication
Malicious Server	manipulate the global model to utilize shared computational power in building malicious tasks	• Compromised Server
Communications Bottlenecks	communication bottlenecks can disrupt the FL process and techniques that aim to reduce the communication cost, such as compression, can be used in a harmful way to introduce noise in the model updates and degrade their quality	• Communication • Distributed Nature
Free-Riding Attacks	dissimulate participation in the FL process with the goal of obtaining the final model without contributing to the training process	• Compromised Clients
Man-in-the-Middle Attacks	intercept the models exchanged between the clients and the server and replace them with malicious updates	• Communication
Dropout of Clients	clients drop out due to network issues, unexpected roadblocks, or otherwise becoming temporarily unavailable, resulting in fairness issues and depriving the model of potentially precious data	• Non-malicious Failure • Communication

model updates can be sorted out of the aggregation process. Robust aggregation algorithms deal with the decentralized nature of the FL approach and sort malicious updates out [46, 42]. Other attacks are addressed by implementing an evaluation score

from all participants to the models being aggregated.

Table 2.2.: Summary of federated learning defenses. Source: [37]

Defenses	Description	Attacks Defended Against
Anomaly Detection	explicitly detect malicious updates and prevent their impact on the system	• Data Poisoning • Model Poisoning • Free-Riding Attacks
Differential Privacy	inject a small amount of statistical noise into the model updates that is empirically sufficient to restrict the success of attacks and minimizes the risk of gradient leakage from the model parameters	• Data Poisoning • Model Poisoning • Inference Attacks • Evasion Attacks
Trusted Execution Environment	a high-level trusted ecosystem for executing attested and verified code while maintaining confidentiality, authenticity, privacy, integrity, and data access rights	• Training Rules Manipulation • Compromised FL Distributed Computation • Malicious Server
Robust Aggregation	aggregation algorithms that can detect and discard faulty or malicious clients during training and sustain communications instabilities, clients drop out, erroneous model updates, and heterogeneous clients	• Non-Robust Aggregation • Data Poisoning • Model Poisoning • Backdoor Attacks • Dropout of Clients
Pruning	reduce the deep learning model's size by dropping neurons to decrease the complexity, improve the accuracy, and disable backdoors	• Backdoor Attacks • Communications Bottlenecks
Zero-Knowledge Proofs	cryptographic primitives used to verify that updates submitted by the clients are conforming to pre-specified properties without sharing or revealing underlying data	• Data Poisoning • Model Poisoning • Backdoor Attacks • Man-in-the-Middle Attacks
Adversarial Training	a minmax optimization problem, where the adversarial samples and the model parameters are alternatively updated	• Evasion Attacks
Federated Multi-Task Learning	learn models for multiple related tasks simultaneously to ensure fault tolerance	• Dropout of Clients • Communications Bottlenecks
Moving Target Defense	randomize FL system modules to reduce the likelihood of successful attacks and shorten attacks lifetime by adding new dynamics to the system	• Inference Attacks • Man-in-the-Middle Attacks • GAN Reconstruction Attacks

3. Related Work

In recent years, research focusing on blockchain technology to address vulnerabilities in the FL approach has become increasingly important. In the following, relevant publications related to the contributions of this work are presented.

3.1. Decentralization

Qu et al. [47] use a permissionless blockchain network as a decentralized curator for FL. In their proposed procedure, once a client has trained its local model, it sends a transaction proposal to all miners, who add the aggregated model to the blockchain with the Proof of Work consensus. One drawback of their implementation is the publishing of local models and aggregated models to the blockchain. This restricts the size of the model that can be used. They also evaluate the use of different values for the deadline time in the training phase. They show that the highest central model's performance is achieved by selecting the training deadline time as high as possible depending on each scenario.

Using a permissionless blockchain can lead to privacy leaks on sensitive information for users when using it in the Internet of Things (IoT). Using the transactions posted on the blockchain, attackers could reconstruct transactions within entities and infer the real identity of the entities. Moreover, data transparency could lead to data being misused by third parties. Since permissionless blockchains have these security disadvantages and low transaction throughput, it is a good practice to use permissioned blockchains in the corresponding use cases. [48]

Sun et al. [49], Y. Li et al. [50] and Desai et al. [51] achieve the decentralization of the curator by storing the local and central models on a permissioned blockchain. In this way, models are available only to authorized users. Rahman et al. [52] improve this aspect while keeping high models' availability and the authentication of the models' integrity by sharing only the model's hash through the blockchain. Models are uploaded through a peer-to-peer hypermedia protocol called InterPlanetary File System[1] (IPFS).

[1] https://ipfs.io/

3.2. Privacy

Several works [53, 52, 54] use differential privacy to address the gradient leakage vulnerability and GAN reconstruction attacks. Rahman et al. [52] and K. Wei et al. [54] evaluate the effect of adding differential privacy on a model's accuracy. They conclude there is a tradeoff between the central model's performance and the privacy protection of local clients. Differential privacy parameters used in this thesis are based on their results.

3.3. Quality Control

The correct behavior of clients has to be ensured to take into account the vulnerability of FL to compromised clients. Additionally, the distributed nature of FL opens the door for compromised clients to collude and carry out coordinated attacks. The use of robust and quality-aware aggregation algorithms plays an important role against this type of attack [38]. Zhang et al. [55] provide an FL platform for device failure detection in industrial IoT using the Ethereum[2] blockchain. They propose an aggregating algorithm that considers class imbalance in the training data set. Data integrity is ensured by creating a Merkle tree of data set hashes. One of the disadvantages is still the need for a central organization, which is the owner of the FL platform. Several approaches implement the selection of a consortium to evaluate uploaded models. The consortium procedure used by Rahman et al. [52] stores the historic reputation score of each client.

Lu et al. [53] provide a Proof-of-Quality consensus mechanism for a permissioned blockchain. In this procedure, a subset of the clients is selected as the committee each round. Each client in the committee tests the model of another committee client and reports their computed scores to the committee leader. Only the committee takes part in the training of local models in each FL round.

Y. Li et al. [50] separate the clients each round into training clients and committee consensus clients. The committee is elected every round based on the score of the clients in the previous round. The committee calculates a score for each model update. They evaluate models with all their available data and take the median of the computed scores for the aggregation. The aggregation is done through the weighted average of the model updates. Using the median as the score for the weighted average makes this approach robust to 50% of the committee's size of malicious colluding clients. On the other hand, Sun et al. [49] evaluate each model update on the endorsing node of the blockchain, which contains a test set. If the computed score reaches a threshold,

[2]https://ethereum.org/

the model will be used to aggregate the central model. A downside is a need for an exposed test set on each endorsing node.

3.4. Incentive Mechanism

An advantage of using a blockchain platform is distributing financial incentives automatically through smart contracts. In the approach proposed by Zhang et al. [55], incentives are calculated based on the size and quality of the data contribution. Kim et al. [56] propose a rewarding mechanism that provides incentives proportional to the training sample size. However, this approach is vulnerable to clients augmenting their data without providing more insights into the central model. Qu et al. [47] also provide rewards linear to the data sample size. However, data size and computation time for the training phase are compared to avoid sample size poisoning to receive a bigger reward. Desai et al. [51] provide a penalty structure in which a client can accuse another client of injecting a backdoor attack in its local model update. The model is verified by their novel attacker detection algorithm against pixel-pattern backdoor attacks. In case the local model is malicious, an amount of cryptocurrency is taken from the malicious client; otherwise, the accusing client loses that amount of cryptocurrency. In this framework, the consortium provides the attacker detection algorithm, and it has to be developed according to the task being solved.

3.5. Accountability and Authentification

Other approaches use blockchains as means for achieving accountability and authentication of users. This targets the exposure of models in the communication channels used. Otoum et al. [57] achieve trustworthy shared training on the fog by implementing a decentralized federated learning process with a blockchain. They avoid data centralization on vehicular networks by implementing a practical byzantine fault tolerance consensus protocol for vehicle identification and authentication in vehicle networks. Kim et al. [56] use client identification for improved accountability. Desai et al. [51] use on-chain aggregation of the local models in Hyperledger Fabric. This provides authentication of the clients uploading local models. The server can verify their credentials and give the authorization to perform the wanted action.

3.6. State of the Art

Mugunthan et al. [12] propose a blockchain-based decentralized FL framework, called BlockFlow. It uses the validation scores computed by all clients against other clients' models to ensure trust between them. Differential privacy is employed against model inference attacks and to preserve the privacy of clients. The local models' hash is stored on the blockchain, and the models themselves are encrypted and shared through the IPFS protocol. The approach leverages an Ethereum blockchain smart contract to provide for client accountability. It is robust against a threat model containing a minority of malicious clients. Each client shares its model, downloads and evaluates other clients' models, posts the evaluation scores to the BlockFlow smart contract, and generates a weighted average central model from all available models and their scores. The score is obtained by querying the median of the reported evaluation scores for each model. The costs of the approach are analyzed and measured in terms of gas, which is defined as 1×10^{-9} Ether, which is the native currency of Ethereum.

The costs of the proposed FL framework are high, and we propose a cost-effective approach using a permissioned blockchain. Additionally, the procedure has to be tailored to the wanted task and model since a deadline time for the training phase has to be given, as in [47]. Therefore, we propose a more robust approach by assuring that a predefined percentage of the available clients participate in the FL round.

4. Hyperledger Fabric Network

In Fabric, entities are structured as organizations. This composition provides a structure that can be used for the predictive maintenance problem. One option is having each machine as a peer. A peer has its database and computational resources, avoiding data centralization. Machines' owners are the organizations that contain an MSP of which machine belongs to them. This concept has a more straightforward configuration on the enterprise's side and would work in a plug-and-play manner. However, one downside is that the approach is less robust against the collusion of malicious clients, i.e., the robustness is only guaranteed against collusion between less than 50% of the peers in the network and not against 50% of the enterprises in the network. Additionally, each machine should have enough computing power to train a local model. The other option is having one edge device per enterprise. Machines belonging to the enterprise send their data to the edge device, which would be the only peer of the organization. It contains enough hardware resources to train the local model with its data. This procedure implies an increased complexity in the configuration on the enterprise's side but provides better robustness against the collusion of malicious clients. We elect this option for the development of this thesis.

Reasons for using a Fabric network are the distributed ledger, increased accountability, member identification, and smart contracts. Having a distributed and synchronized ledger habilitates the decentralization of the whole FL procedure. Information can be shared without having a centralized database, and read/write conflicts can be automatically addressed. The identification of a member's real identity and storing all performed transactions in the ledger provide accountability to the process. Additionally, the authorization for the use of smart contract methods is specified by the consortium of organizations.

Section 4.1 presents the framework created to automatize the configuration and deployment of a Fabric v2.2 network, called Fabric Network Generator. Fabric Network Generator takes care of crucial stages such as the network's deployment, joining organizations to a channel, submitting a chaincode to a channel, and installing the chaincode in the peers connected to the channel. Thereafter, section 4.2 provides insights to the generation of the configuration files used by Fabric v2.2 components.

4.1. Fabric Network Deployment

The Fabric Network Generator was developed to easily automatize the necessary and complex steps of intertwining a Hyperledger Fabric network.

A configuration file detailing the components of the organizations and their attributes is given to the framework. The network gets its topology from this file. Additionally, the path to the chaincode source code has to be given. The final output is a network containing one channel to which all peers are joined. The smart contract has been committed to this channel, and installed on each peer.

Going further into the procedures taken by the framework, a distinction between several stages can be made. First of all, the backbone of each organization is constructed. Here, a docker container for the organization's certificate authority is initialized. This component contains the certificate, which will sign all other organization components. The signing of components makes it possible for other components to verify to which organization a component belongs. It also contains the Transport Layer Security (TLS) certificate used for the network's communication protocol. Next, an administrator user is created, and some configuration about the file-storing structure is given. In this step, the organization administrator is also registered and enrolled in the CA. After the certificates and administrator are ready, the components are created. Each component has to register its name, password, and type to the organization's CA. After that, they enroll and obtain a TLS certificate, a signed certificate from the CA, and a private key. After the components are ready, the organization gathers all of the components' certificates. They are used to create a Membership Service Provider for the whole organization. Finally, the components' docker containers are brought up.

A Fabric-Client Docker container is brought up in the last part of this stage. This container contains all the binary files used to configure and communicate with the network's components. Their parameters given to establish a communication with an organization's component are:

(i) The organization's MSP ID

(ii) The organization's MSP Path

(iii) The component's IP address

(iv) The component's TLS root certificate

The second stage creates and joins components to a channel through which they will communicate later. The first step in this stage is the creation of the channel. A configuration profile is used to create the channel's genesis block. This Block is the

starting point for all the ledgers belonging to this channel. Additionally, it contains all the configurations and policies used for applications and authorization. Finally, the genesis block and channel ID are given to the ordering service that manages the distribution of transactions in this channel. The second step is the joining of the components to the channel. Each peer is given the channel's genesis block. Then, they contact the ordering service and synchronize their ledger with the one shared over the channel.

The third stage installs the chaincode in all the peers available in the channel. First, the chaincode must be packaged and installed in at least one peer per organization. This peer uses the organization's administrator to approve the commitment of the chaincode to the channel. For the chaincode to be committed in the channel, the chaincode commitment policy defined in the channel's configuration must be met. In our case, the policy states that all organizations' administrators have to review and approve the chaincode. Then, once all organizations have approved the chaincode, it is committed to the channel. This starts an additional chaincode Docker container for each peer having the chaincode installed. Finally, the remaining peers can install the chaincode and start their own chaincode Docker container. The Software Development Kits (SDK) provided by Fabric can be used by applications to create a transaction using the chaincode methods.

4.2. Fabric Network Configuration

In addition to deploying the components and creating the network, the Fabric Network Generator also takes care of the formulation of the configuration files. These can be separated into four categories: (i) Docker Compose files, (ii) Channel Configuration file, (iii) Fabric CA Server Configuration file, and (iv) Connection Profile files.

(i) Each component has a Docker Compose file. The framework contains templates that have the basic requirements that each component needs. The networking configuration and other environment variables are given to each component depending on the network's topology.

(ii) The Channel Configuration states which organizations are part of the channel, the ordering service type, and the orderers available at the beginning of the channel's existence. To add an organization afterward, the consensus of the corresponding policy has to be satisfied.

(iii) Each organization has its Fabric CA Server. It must have the corresponding attributes of that organization. It mainly contains the configuration of the or-

ganization's CA, signing attributes used to sign enrollment certificates, and the configuration given for the certificate signing request of the root CA certificate.

(iv) Connection Profiles are used by applications using the provided Fabric SDK.

The number of CPU cores and memory used by each component can be limited directly on the Docker Compose files. The CPU limits given are soft limits that the docker container can overpass for a short period. The memory limits are hard limits, which means the container will be stopped once it surpasses the given limit. This is to be taken into account in the designing of the components.

The world-state is stored in each ledger of the peers that joined the channel. It can be modified by the methods defined in the chaincode. Figure 4.1 shows an example of what a world-state with only two clients looks like. The prefix of each key defines the attribute being selected and the suffix the client's ID. This way, the reading of a specific attribute for all clients can be efficiently done through a range query giving the attribute as start key and the attribute plus a "~" sign as the end key. In the ASCII table, the tilde sign can be translated into the decimal number 126, which is the last writable character. The chaincode also provides methods to create clients, delete clients, modify single client's attributes, and do range queries for a specific attribute or client.

Figure 4.1.: Example of the ledger's world-state containing two clients.

To make the chaincode methods available for a python application, a REST API Server of the Chaincode was made using the fabric-network library available for

Javascript. A Python SDK is also provided by Fabric but has currently little functionality implemented.

5. HyperFlow

In this section, we propose an approach that uses the Hyperledger Fabric smart contract platform to address the vulnerabilities of FL approaches in terms of compromised clients, compromised servers, gradient leakage, aggregation algorithm, and distributed nature in a cost-effective process we call HyperFlow. It is a cost-efficient framework based on the state-of-the-art procedure BlockFlow [12]. HyperFlow has been designed modularly so that new ML models and data loaders can be easily included. Additionally, the framework extends the state-of-the-art by implementing time-independent deadlines for each phase of the FL approach. Deadlines are configured as a given percentage of clients that should be available to proceed to the next phase.

Section 5.1 presents a detailed description of a HyperFlow client's components. Furthermore, Section 5.2 introduces the workflow of the FL approach.

5.1. HyperFlow Client

The HyperFlow client manages the training of a local model, the phases' synchronization between clients, and the aggregation of the central model. Differential privacy is used as a privacy-preserving mechanism for training local models. In the usual setting of FL, the curator is in charge of providing clients with signals to synchronize the FL approach, aggregating the central model and deploying it to the clients. The curator's decentralization provides robustness against a compromised server, but these tasks have to be accomplished by an entity. Each HyperFlow client executes these tasks for itself. It employs a percentage strategy for the synchronization of the FL phases. The local models and aggregation score are available to each client to aggregate the central model. At any point, a client can drop out. This will mean the most recent central model will not be available to this client. Moreover, suppose that the client cannot download one of the other models for any reason, e.g., firewall configuration or internet problems. In that case, the computed central model will not concur with the overall central model. This issue is solved by synchronizing the central model over clients at the start and end of each FL round. The synchronization procedure is shown in more detail in Section 5.2.

Figure 5.1 shows the components of a HyperFlow client. The Cloud Client takes care of the uploading and downloading of models. The Crypter manages the encryption and decryption of objects, while the Blockchain Connector submits requests to the smart contract's Node REST Service. The Deadline Manager sets the strategies for the client's synchronization. The Data Loader communicates with the Database and processes data to train an ML model. The Deep Learning Module configures the ML model to be used and applies a privacy-preserving method to it. A YAML file gives the configuration. An example can be seen in Appendix A. The configuration is loaded using the Dynaconf[1] Python library. This way, configuration variables can also be given as environment variables.

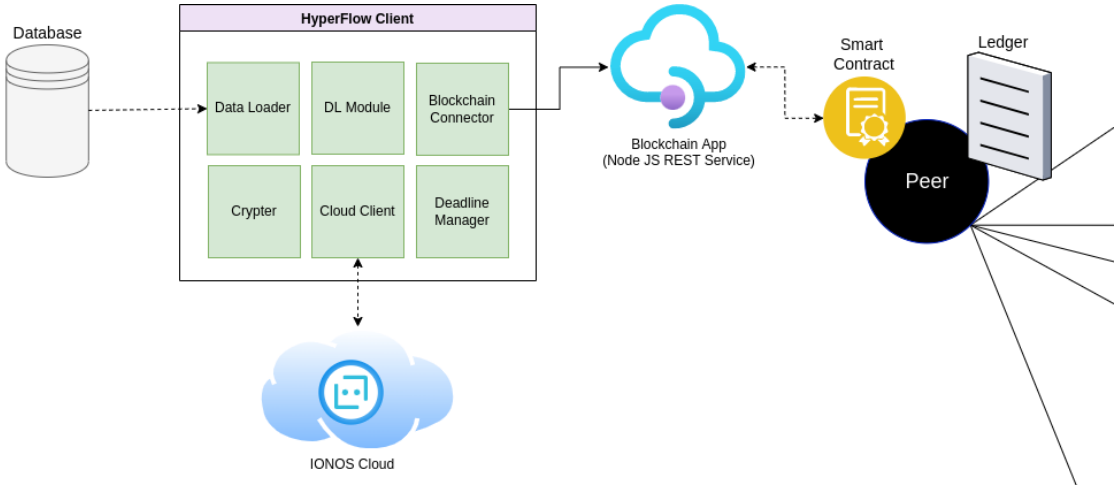

Figure 5.1.: Components of a HyperFlow client.

Subsequently, each component will be presented in a detailed manner.

5.1.1. Cloud Client

The HyperFlow client uses the S3 file storage system to store weights for the trained local models and make them available to all other clients. The storage cloud used for the approach is the IONOS Cloud. For this, a bucket was set in the IONOS S3 File Storage System. The access authorization to the S3 bucket is settled through the access key and secret key given in the configuration file. Then, using the adequately generated headers, the cloud client can send requests to upload or download a file. The headers are generated using the AWS4-HMAC-SHA256 algorithm.

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

5.1.2. Crypter

The Crypter acquires the task of encrypting and decrypting the evaluation metrics. They are encrypted using symmetric encryption. A new encryption key is generated and posted in the corresponding phase to the blockchain in each round.

5.1.3. Deep Learning Module

A degradation model is used to calculate a component's remaining useful lifetime (RUL) in predictive maintenance. Each Hyperflow client has its degradation model. It is set as an independent module and is defined through the configuration file. The model used for the evaluation is an LSTM model. The configuration file gives attributes such as the sequence length, LSTM layers, neurons per layer, dropout rate, activation function, and several other parameters. The created model is a Keras model with TensorFlow v2.7 as the backend. The early stopping is configured and guaranteed through a Keras callback function which monitors the validation loss. The loss used is the root mean square error between the predicted RUL and the target RUL. Additionally, the implemented module provides means to facilitate the aggregation of weights from several models. The hash generation of the model's weights is done using the SHA256 hashing algorithm. The used parameters and training and validation curves are logged using the MLFlow framework. This way, monitoring the curves over multiple federated learning rounds is possible. Differential privacy is added using the Differential Privacy Keras Adam Optimizer. Parameters for the Differential Privacy Optimizer have to be given through the configuration file. These are the l2 norm clip, the noise multiplier, and the number of micro-batches.

5.1.4. Data Loader

The Data Loader contains methods for generating training, validation, and testing data. In this work, the extraction, processing, and loading of data from the TurboFan Data Set [16] is implemented. The data set and its pre-processing methods are further explained in Section 6.1.

5.1.5. Blockchain Connector

The Blockchain connector sends requests to the blockchain's REST API Server. Each HyperFlow client creates their application user in the blockchain. This allows them to modify only their own attributes in the blockchain's world-state. Additionally, range queries are used to get attributes of the same type efficiently.

5.1.6. Deadline Manager

The deadline manager manages the synchronization of clients on each phase. There are two deadline strategies the deadline manager can take. The first one is the percentile deadline. In this case, the threshold percentage parameter has to be given. Once the number of clients in the same phase reaches this threshold, all clients continue to the next phase. The clients who did not reach the phase in time are not considered. The second deadline type is the hard-time constrained deadline. The time constrain for each phase is given. Once the time for a phase has elapsed, the clients move to the next phase. In this deadline strategy, the synchronization of clients participating in the federated learning round has to be assured.

5.2. Federated Learning Workflow

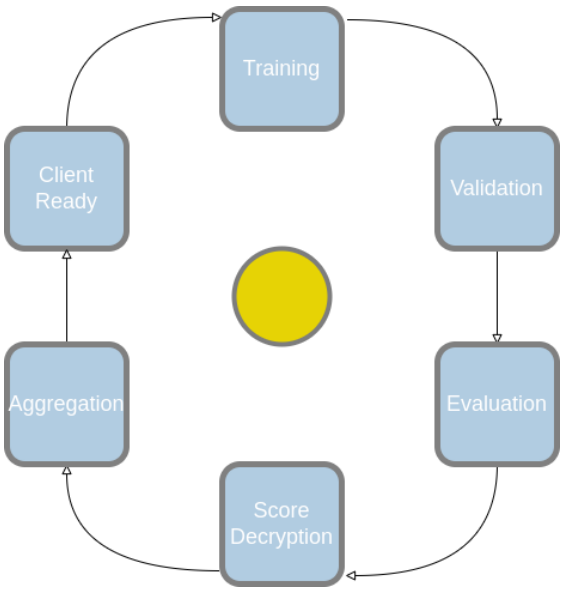

Figure 5.2.: Overview of the phases in a federated learning in HyperFlow.

An FL session consists of several rounds where clients train local models and aggregate them into a central model. In this section, the phases in an FL round are explained in detail. Figure 5.2 presents a high-level overview of one FL round that is made up of six phases:

1. Client Ready

2. Training
3. Validation
4. Evaluation
5. Score Decryption
6. Aggregation

5.2.1. Phase 1: Client Ready

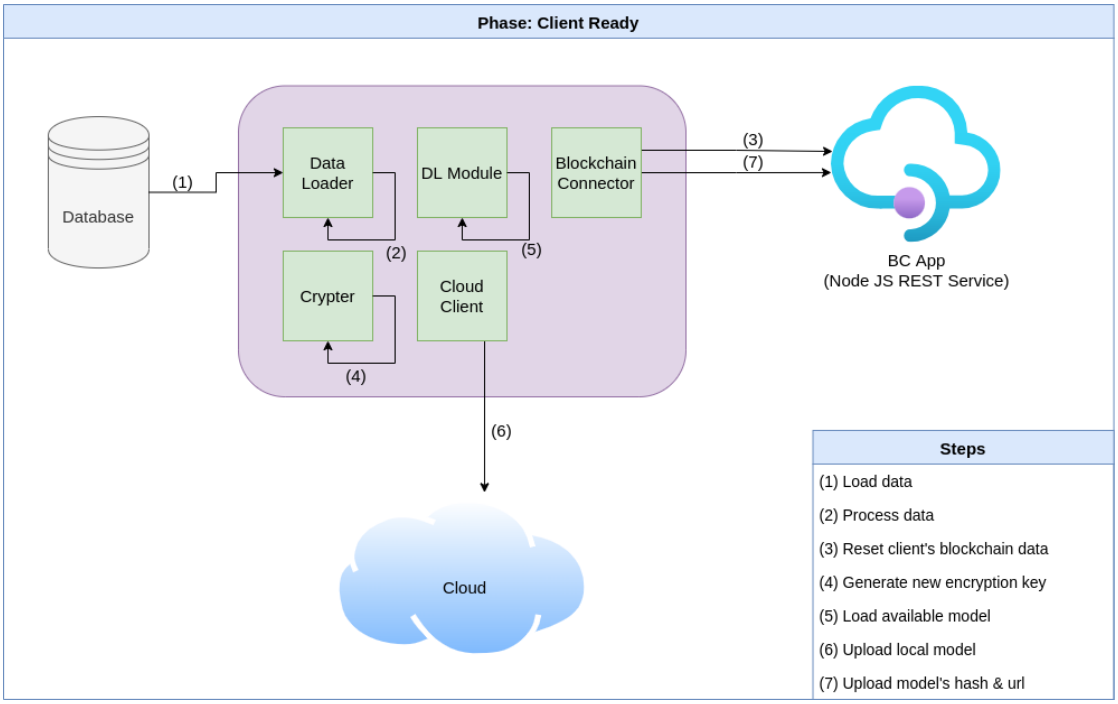

Figure 5.3.: Workflow of the Client Ready phase.

The first phase is the Client Ready phase. First, data registered from the client in the blockchain's world-state is erased. Second, the client loads the data using the Data-Loader module described in Subsection 5.1.4. In the case that weights are available to the client, the Deep Learning Module, described in Subsection 5.1.3, loads the model available. Otherwise, the Deep Learning Module initializes a Keras model as specified in the configuration file. Later, the Crypter, described in Subsection 5.1.2, generates a new key that is later used to encrypt the computed evaluation metrics. Afterwards, the

Cloud Client, described in Subsection 5.1.1, uploads the current model. Finally, the Blockchain Connector, specified in Subsection 5.1.5, receives the hash and the cloud URL of the model and registers them in the blockchain. The workflow of the Client Ready phase is shown in Figure 5.3.

5.2.2. Phase 2: Training

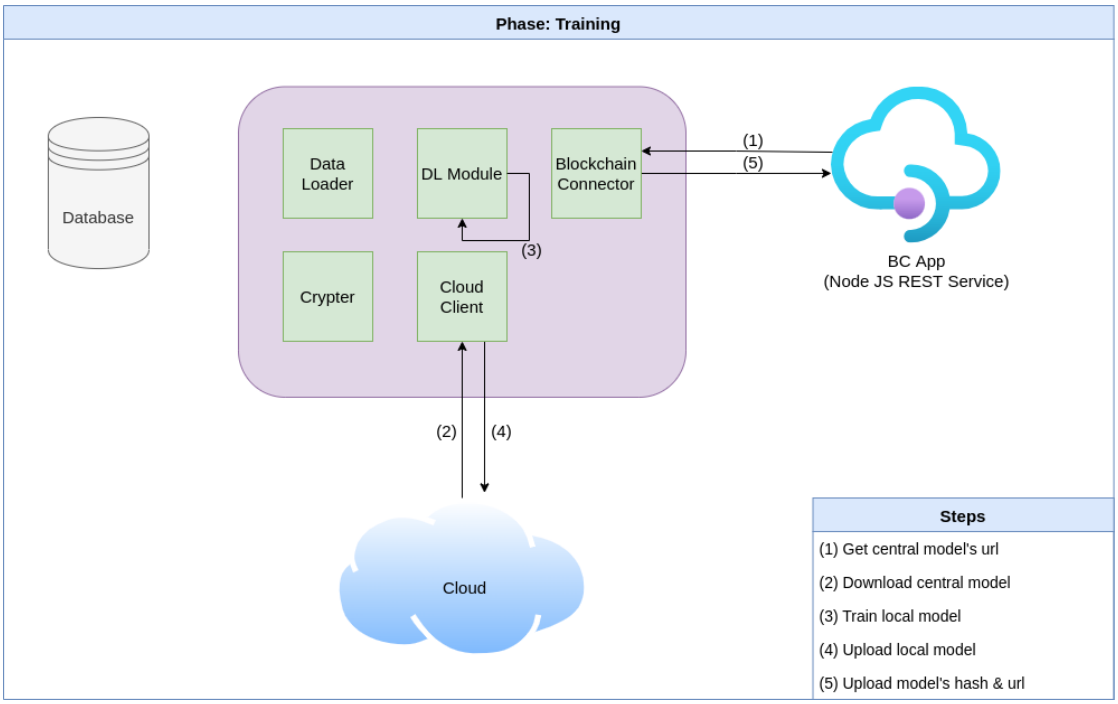

Figure 5.4.: Workflow of the Training phase.

The second phase is the Training phase. First, the central model is synchronized over the available clients. Then, each client uploads its current model to the cloud, and registers its hash and URL to the blockchain. Since we propose that the approach is robust against a minority of compromised clients, the central model will be determined by more than 50% of clients having the same aggregated central model. The comparison is made over the registered hashes. Clients having a different model will download the central model of the majority of clients. In case there is no congruent model over the majority of clients, each client keeps their model. These conditions are common when initializing the model for the first time. Afterward, the model's training and evaluation are done using the methods provided by Keras. The history of the training and

evaluation curves are uploaded into the MLFlow Framework. Additionally, parameters used for the training and differential privacy are stored. After the model training is finished, the Cloud Client takes over the task of uploading the trained model, and the Blockchain Connector registers the download URL and the model's hash to the ledger's world-state. The Training phase's workflow is shown in Figure 5.4.

5.2.3. Phase 3: Validation

The Validation phase prepares the pre-selection of models taken into account for the aggregation of the central model. Each client tries to download all models. Validation flags are then set for each model the client could successfully download. After all available models are validated, the validation flags are registered in the blockchain. This workflow is presented in Figure 5.5.

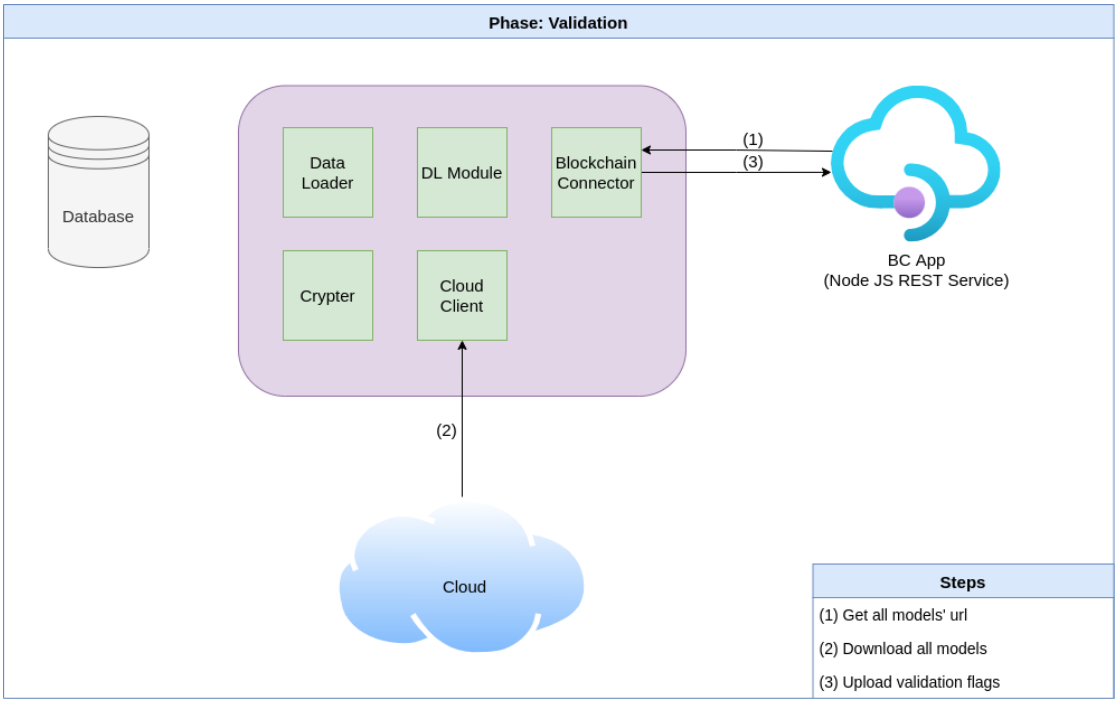

Figure 5.5.: Workflow of the Validation phase.

5.2.4. Phase 4: Evaluation

The pre-selection of models is done by calculating the model's validation score as seen in Equation 5.1, where C is the set of available clients in the FL round, M the set of

available models, $v_{i,j}$ the validation flag given by client i for model j, and n the number of available clients in the FL round. To maintain the robustness to compromised clients of the approach, only models available to 100% of the participating clients are evaluated. Afterward, the metrics computed are encrypted and registered in the blockchain. This phase's workflow is presented in Figure 5.6.

$$\forall_{j \in M} score_j = \frac{\sum_{i \in C} v_{i,j}}{n} \tag{5.1}$$

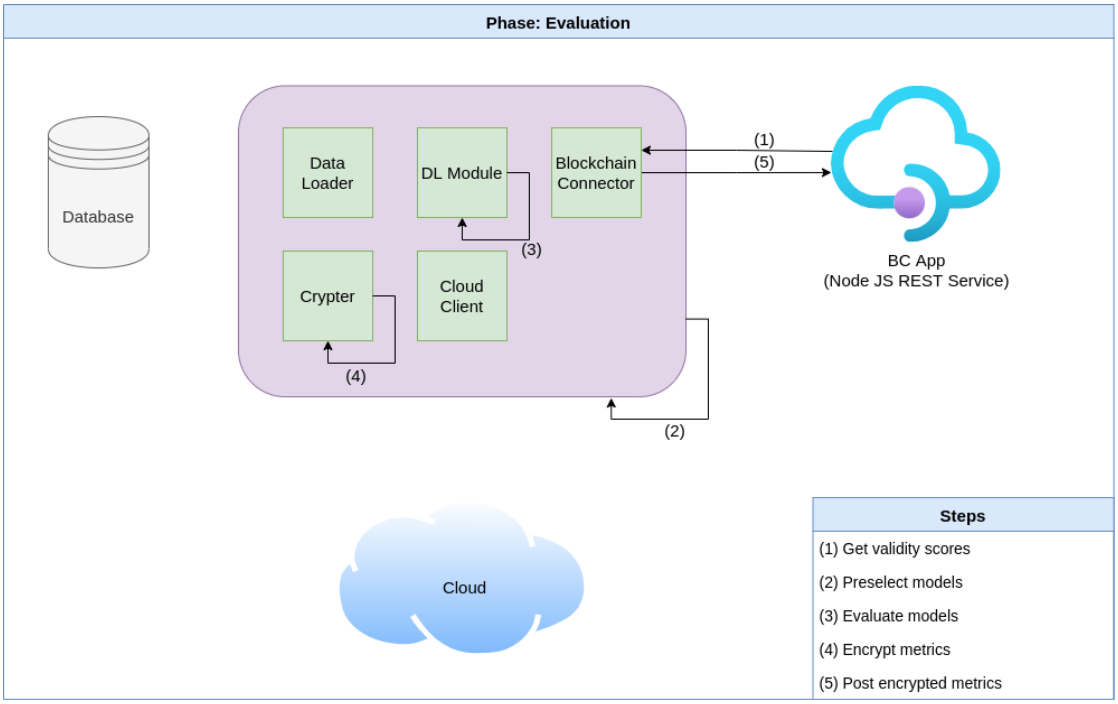

Figure 5.6.: Workflow of the Evaluation phase.

5.2.5. Phase 5: Score Decryption

In the Score Decryption phase, the key used for the symmetric encryption of the evaluation metrics is registered in the blockchain. This makes the decryption of all evaluation metrics available to all clients. This phase's workflow is seen in Figure 5.7.

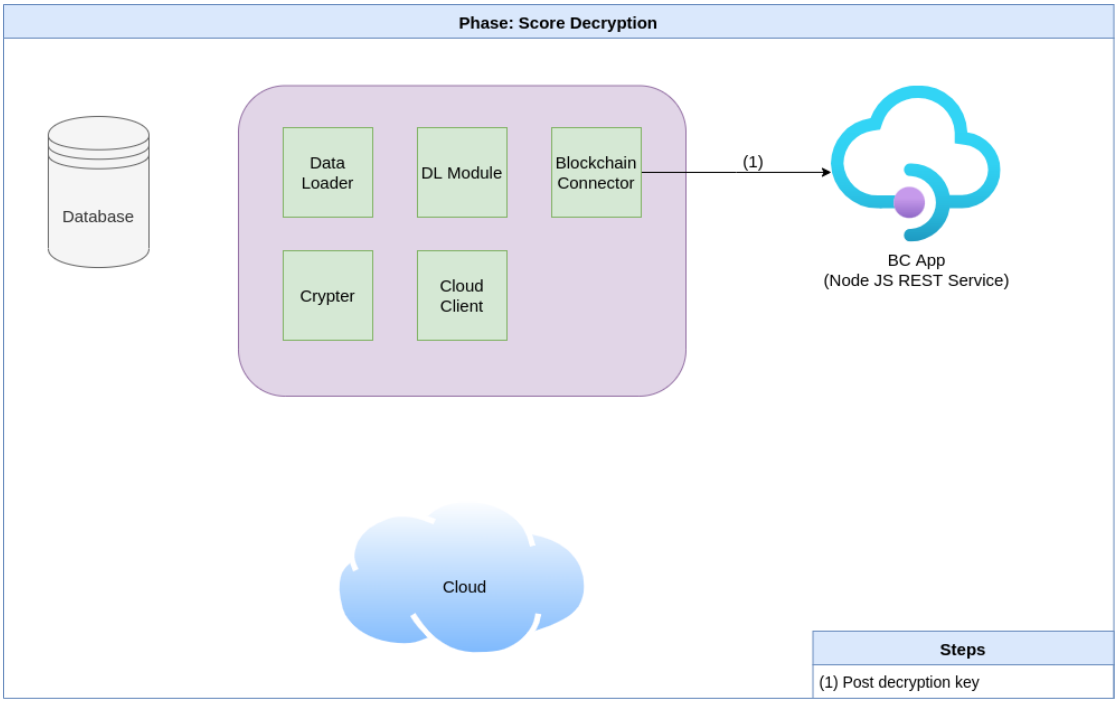

Figure 5.7.: Workflow of the Score Decryption phase.

5.2.6. Phase 6: Aggregation

In the Aggregation phase, all evaluated models are joined. The central model produced is a weighted average of the clients' models.

The aggregation scores are computed as shown in Equation 5.2, where $\tilde{M}$ is the set of pre-selected models, w_j the aggregation score for model j, C the set of participating clients, and $\theta_{i,j}$ the evaluation score of model j computed by Client i. Thereafter, the scores are scaled using the maximum weight as a reference as stated in Equation 5.3.

$$\forall_{j \in \tilde{M}} w_j = median(\forall_{i \in C} \theta_{i,j}) \tag{5.2}$$

$$\tilde{w}_j = \frac{w_j}{\max \forall_{i \in \tilde{M}} w_i} \tag{5.3}$$

An Apriori of 0.5 is selected, i.e., models having an aggregation score below half the score of the best performing model in the round are not considered for the aggregation. The score for models not been considered for aggregation is directly matched to zero, and normalization is done over the remaining scores. The aggregation scores become robust against the collusion of malicious clients by selecting the median of

the evaluation scores. A minority of malicious clients implies that the median of the evaluation scores lies in the set of benign clients' computed evaluation scores. The distinction between publishing and revealing the evaluation metrics is important to avoid malicious clients targeting their evaluation score near the aggregation score. Afterward, the aggregated model is uploaded and registered in the blockchain. Finally, the central model is synchronized over clients. The final synchronization is done in case a client got hung up on computation or had a connection issue. The workflow in this phase is shown in Figure 5.8.

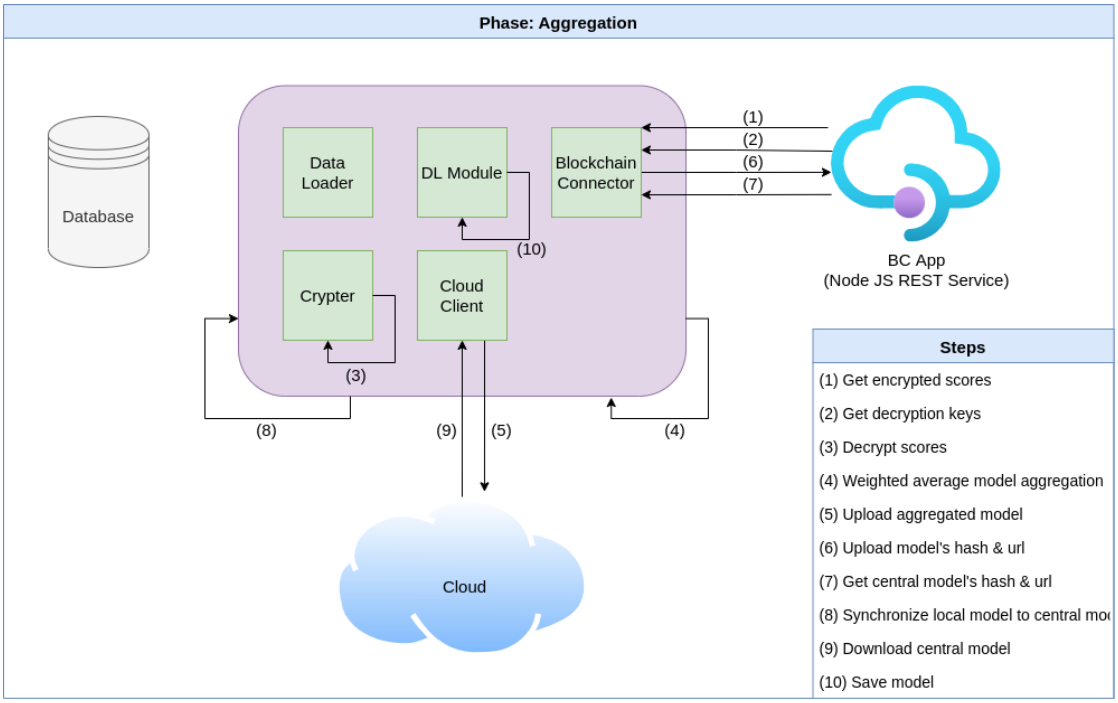

Figure 5.8.: Workflow of the Aggregation phase.

6. Experimental Setting

This chapter presents the experimental setting used to evaluate the federated learning characteristics of the proposed approach. First, the data set used to evaluate the HyperFlow approach is presented. Later, the deep learning models analyzed for the predictive maintenance task are described, and the selection of the model used for further evaluation is reviewed.

6.1. Data Set

The Turbofan Engine Degradation Simulation Data Set [16] is used to benchmark different approaches of RUL predictions [58, 59, 60]. It was produced using the NASA Commercial Modular Aero-Propulsion System Simulation[1].

The data set is composed of four subsets and simulates the life cycle of turbofan engines. Each subset contains a training and a test set of multiple multivariate time series. The time series simulate different turbofan engines with distinct initial characteristics such as the degree of initial wear, operating conditions, and manufacturing variations. The initial wear is simulated by small initial variations in the flow and efficiency factors of the engines' modules. Additionally, the main sources of noise, e.g., assembly variations, process noise, and measurement noise, are modeled by introducing normal-distributed noise from a mixture of Gaussian distributions in different stages of the simulation process. [16]

Each data point in a time series consists of the engine-ID, 21 sensor values, and three operating conditions. The training set and testing set have different structures. Engines in the training set operate normally at the beginning and acquire a fault at some point in the time series. Thereafter, the fault develops until a system failure occurs. The test data set provides data of engines until a point in the time series before the system failure. This data is used to measure the accuracy of the trained model to predict the RUL of the engine.

The subsets vary in the number of operating conditions the engines run and the faults appearing in the engines. Subsets FD001 and FD003 contain only one operating condition, while subsets FD002 and FD004 contain six of them. Only one fault condition

[1] https://software.nasa.gov/software/LEW-18315-1

appears on the engines of subsets FD001 and FD002, while two fault conditions appear in subsets FD003 and FD004. The amount of data on each subset and its characteristics are shown in Table 6.1.

Several sensors remain constant in specific subsets, e.g., 1, 5, 6, 10, 16, 18, and 19 in FD001 and FD003. Therefore, they could be taken out of the feature vector provided to the ML model when building a model for a specific subset. Since we build one model that addresses all four subsets simultaneously, we opt to use all sensor values in our feature vector. Additionally, labels of the training data are clipped to a maximum of 125 cycles as RUL. The three operating conditions parameters are combined in order to use the operating condition as a class attribute. We standardize the sensor values depending on the operation conditions by removing the mean and scaling to unit variance.

Table 6.1.: Summary of the characteristics of TurboFan data set's subsets FD001, FD002, FD003 and FD004.

Data set	FD001	FD002	FD003	FD004
Number of engines	100	260	100	249
Training samples	20631	53579	24270	61249
Test samples	100	259	100	248
Fault conditions	1	1	2	2
Operating conditions	1	6	1	6

6.2. Predictive Maintenance Model

This section presents the model used to evaluate the proposed FL approach in predictive maintenance. Furthermore, the model is evaluated with the test subsets of the TurboFan data set.

The contribution of this work focuses on integrating a permissioned blockchain as a solution to several vulnerabilities of FL. Models commonly used for the prediction of the RUL of the TurboFan data set are Long-Short Term Memory (LSTM) models [59, 61, 62, 63, 60]. In order to define a well suitable model for our federated learning approach, we analyze LSTM models containing one and two LSTM layers together with one fully connected layer.

Table 6.2 shows the parameters used on the hyper-parameter search. The models were trained with the combination of all four available subsets and reviewed on the test subsets. The training of the models was done using an Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz with 8 cores. The average training time was 20.7 minutes.

Table 6.2.: Summary of parameters used on the optimization of hyper-parameters for the LSTM model.

Name	Parameter list
Epochs	30
Learning rate	[0.05, 0.01, 0.005, 0.001]
Dropout	0.1
Sequence Length	30
Nodes per Layer	[128, 256]
LSTM Layers	[1, 2]
Optimizer	Adam
Early stopping patience	5 epochs

Table 6.3.: Results of the hyper-parameter optimization for the LSTM model.

LSTM Nodes per Layer	Learning rate	FD001	FD002	FD003	FD004	Mean
256, 256	$5 * 10^{-4}$	17.31	27.09	17.01	28.52	22.48
	$1 * 10^{-3}$	15.26	25.28	14.86	26.11	20.38
	$5 * 10^{-3}$	17.18	25.89	19.74	26.02	22.21
	$1 * 10^{-2}$	43.63	54.32	43.54	54.57	49.01
128, 128	$5 * 10^{-4}$	15.98	25.20	18.08	27.51	21.69
	$1 * 10^{-3}$	16.85	25.27	17.83	25.70	21.41
	$5 * 10^{-3}$	15.74	25.10	19.01	27.89	21.94
	$1 * 10^{-2}$	43.26	54.15	43.16	54.53	48.78
256	$5 * 10^{-4}$	16.28	27.05	17.85	29.13	22.58
	$1 * 10^{-3}$	13.91	24.78	13.63	25.77	**19.52**
	$5 * 10^{-3}$	13.43	25.84	14.95	26.62	20.21
	$1 * 10^{-2}$	14.51	25.21	15.44	26.22	20.35
128	$5 * 10^{-4}$	16.92	25.55	20.31	27.72	22.63
	$1 * 10^{-3}$	15.61	25.36	18.54	26.35	21.47
	$5 * 10^{-3}$	15.30	25.51	17.24	26.54	21.15
	$1 * 10^{-2}$	17.97	27.16	18.07	27.04	22.56

The results of the hyper-parameter search are shown in Table 6.3. It presents the performance of each reviewed model on each of the test subsets and their computed average. Smaller models containing one LSTM layer tend to be slightly better than models containing two LSTM layers.

The best performing model was a one-Layer LSTM network summarized in Table 6.4. Therefore, this model is used further for all subsequent evaluations.

Table 6.4.: Summary of the best performing LSTM model.

Layer	Output Shape	Trainable Parameters
Input Layer	(30, 14)	0
LSTM Layer	(256)	277504
Dropout Layer	(256)	0
Dense Layer	(1)	257

7. Results and Discussion

This chapter presents the evaluation of the proposed HyperFlow approach. The results obtained are shown and discussed. First, the design and costs of the Hyperledger Fabric network are analyzed. Second, the FL performance in the HyperFlow approach is evaluated. Thereafter, the addressed vulnerabilities for the FL approach, such as privacy concerns and robustness against malicious clients, are reviewed. Finally, the last section evaluates the policies for the percentage deadline criterion in FL.

7.1. Hyperledger Fabric Network

This section analyzes the resources used by the Fabric network for the HyperFlow approach. The evaluation done is independent of the FL use case and ML model used. The necessary resources for the network to communicate with a predefined amount of clients are examined. A benchmark of the network is done on Subsection 7.1.1. Moreover, the resources used in an FL session by the blockchain are recorded and reviewed on Subsection 7.1.2. We use these results to provide comparable blockchain costs of thriving HyperFlow against the State-of-the-Art BlockFlow, which uses the Ethereum Platform.

7.1.1. Network Benchmark

The networks' benchmark was done using the Caliper[1] framework. The procedure simulates the transactions submitted between clients in an FL round. It measures the capabilities of a network having limited resources such as CPU cores and RAM. We analyze specifically the transaction throughput and the failed transactions in a network with a different number of clients. In this way, we can define viable limits for the network's components and evaluate their used resources. The CPU limit is a soft limit, which can be surpassed for a short period. The memory limit is a hard limit, which means exceeding it will terminate the component. A total of 17 types of transactions are simulated and are categorized into write transactions, which update data in the ledger's world-state, and read transactions, which only read data from the ledger's world-state.

[1] https://hyperledger.github.io/caliper/

The first benchmark was done in networks of three, five, and ten clients, where the CPU usage of peers and orderers was limited to 100 CPU milicores, and the memory usage was restricted to 250 MB. Transactions were sent for 10 seconds with a sending rate of 100 transactions per second (tps). The results are shown on the left side of Figure 7.1. The overall transaction throughput decreases with the number of clients in the network. By ten clients, it decreases to 3 tps, and the amount of failed transactions increases to almost 1%. This provides an insight into a shortage of resources on the peers or the orderers. In addition, the failed transactions reported an event service timeout error, which implies that not enough peers could endorse the transaction within the timeout period.

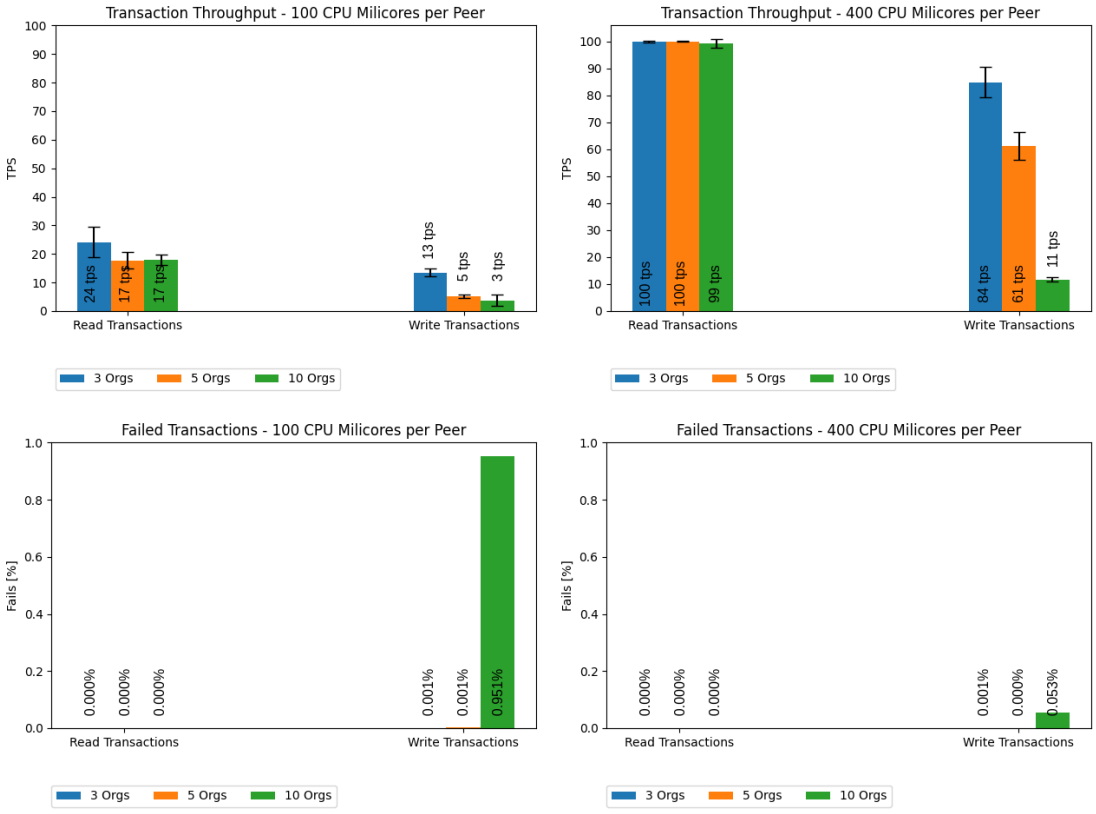

Figure 7.1.: Results on transaction throughput and failed transactions for the analysis done with the Caliper Benchmark Framework.

Increasing the CPU resources per peer to 400 CPU milicores increased the network's performance, as seen on the right side of Figure 7.1. In contrast to the benchmark results with 100 CPU milicores as the limit, failed transactions stay under 0.06%. Additionally,

the network with ten clients processes 11 tps, which is an acceptable rate for the HyperFlow approach containing ten clients. On average, the first transaction submitted fails due to service discovery errors.

7.1.2. Resource consumption

In order to analyze the consumption of CPU and memory resources used by the Fabric network, an FL session containing ten rounds was modeled. Since the focus is not on the performance of the produced models, the training iterations were set to one epoch. Data was collected using the Docker SDK for Python and downsampled to a sampling frequency of 0.1 Hz.

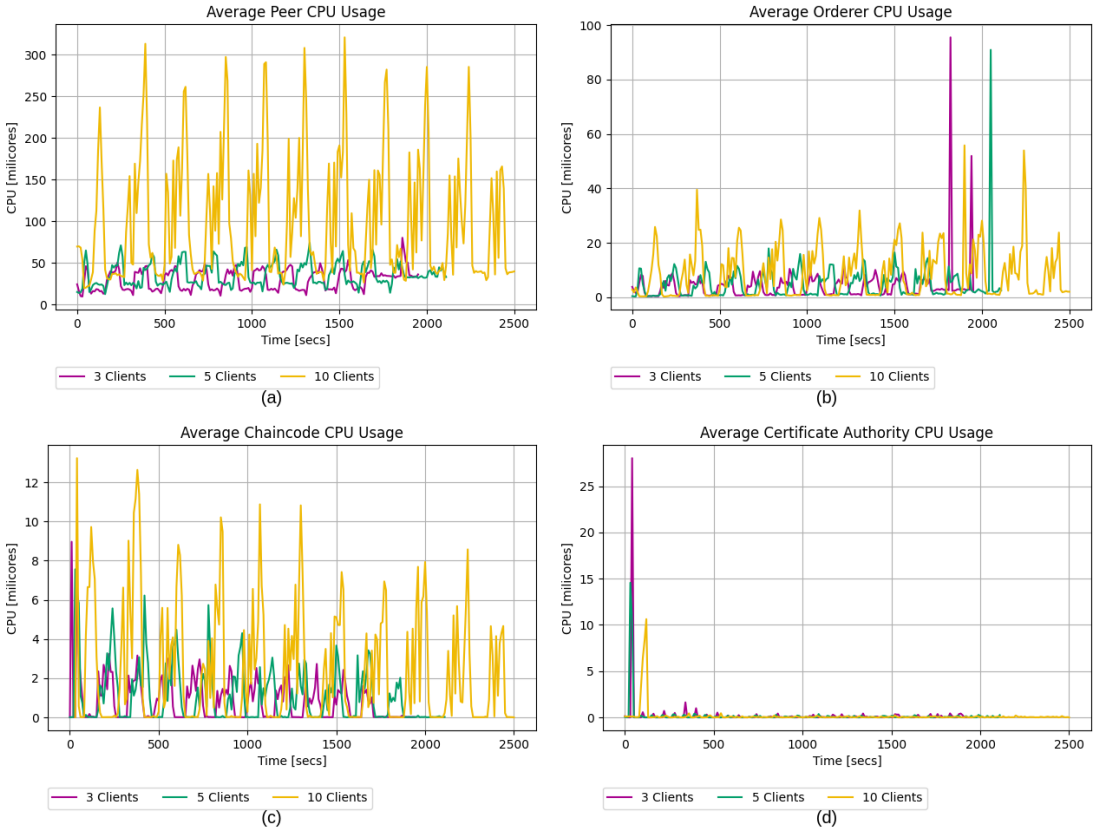

Figure 7.2.: Average CPU consumption per Fabric component.

Figure 7.2 shows the average CPU usage in cores for the peer, orderer, chaincode, and certificate authority components of Fabric. The usage is plotted over time and for a different number of clients present in the FL approach. The HyperFlow client

communicates with the blockchain constantly for reading and updating the ledger's world-state. Therefore, we can match the training of the local model to the valleys on the CPU load of the peers, orderers, and chaincode components. Figure 7.2 (a) shows the exponential increase of workload on peers by the linear increase of clients. This is seen at the peers' CPU consumption curve when comparing the increase in consumption in a network of five clients vs. a network of ten clients. The peak of CPU usage for the five clients' curve is around 60 milicores, while the ten clients' curve has its peak of around 300 milicores. This is related to each peer having to validate and endorse the transactions submitted by all other peers. Looking at the internal working procedure of a peer, we see that validating a transaction means checking all signatures against all peers' certificates to figure out if the signatures are valid. Furthermore, the CPU usage of the orderer, presented in Figure 7.2 (b), maintains relatively low since it only packages the transactions and forwards them to the peers. In the same way, the CPU usage of chaincode components, shown in Figure 7.2 (c), is relatively low and rarely surpasses the ten milicores. The certificate authority works to verify components belonging to a specific organization. This is done at the beginning of the blockchain's application. After that, as seen in Figure 7.2 (d), the workload of the certificate authority decreases to under one milicore.

The average RAM usage per Fabric component is shown in Figure 7.3.

Figure 7.3 (a) presents the average peer's memory. On the one hand, it shows a similar memory consumption for an FL session containing three clients and five clients. On the other hand, an increase in memory usage is seen by the curve of ten clients. The constant increase in the memory allocation of the peer components over time is due to the gossip protocol between peers. The gossip protocol reduces the load an orderer receives by allowing other peers to pull blocks from a peer of the same organization. This implies that peers will stack blocks on their cache to provide them to peers trying to pull blocks through the gossip protocol.

Figure 7.3 (b) presents the average orderer's memory usage. It shows an increase in the orderer's memory consumption over time. The orderer puts transactions together in blocks and distributes them to the peers available. It remains with blocks in the cache so that any new peer connected will receive the blocks to synchronize its ledger to the one of the channel. The number of blocks created is not proportional to the number of clients or processed transactions. The maximum amount of transactions packed in a block is set to ten, and the batch timeout is set to two seconds. This means the orderer will wait for a maximum of two seconds to fill the block with as many as ten transactions. If the two seconds run out before ten transactions are submitted to the orderer, the block will still be sent to the peers to endorse it. This explains the slight increase of memory on the orderer for more clients submitting transactions.

On Figure 7.3 (c), the average chaincode's memory usage is presented. A slight

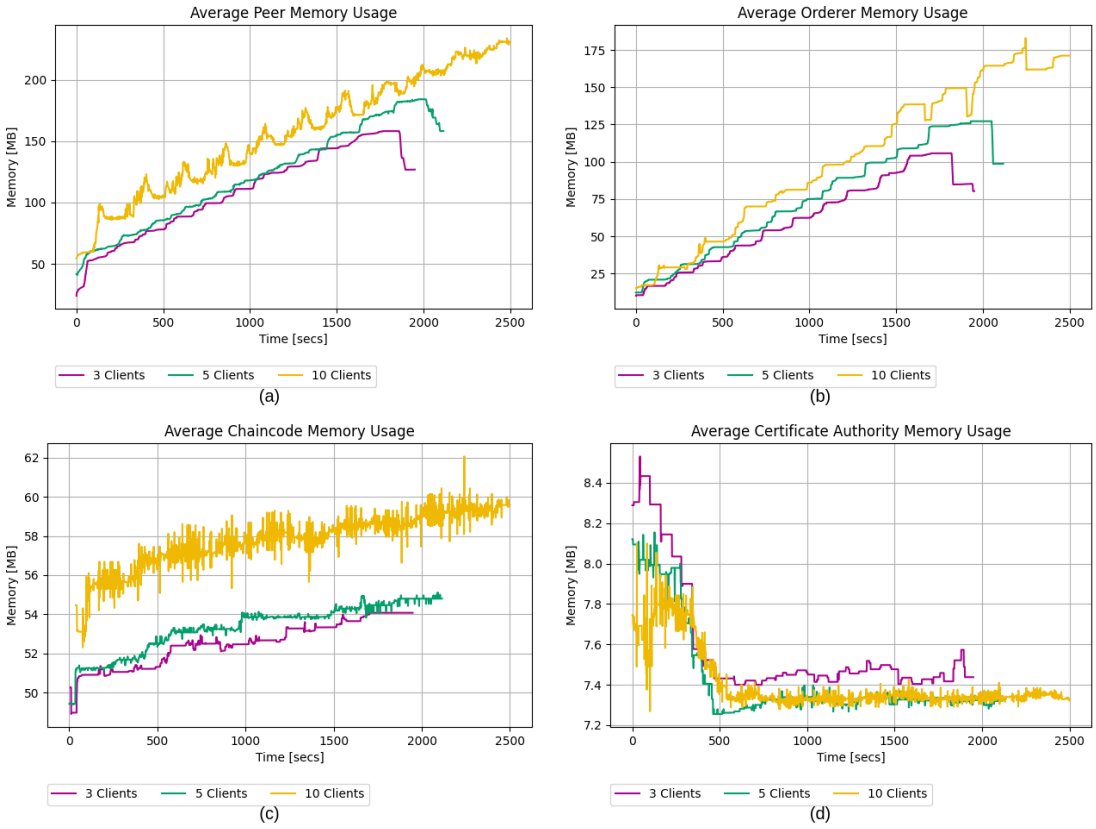

Figure 7.3.: Average RAM consumption per Fabric component.

increase in the average chaincode memory usage is seen over the FL session. On Figure 7.3 (d), the average certificate authority's memory usage is presented. The memory used by the certificate authority component remains constant over the FL session after an initial period.

7.1.3. Costs

Equation 7.1 shows the cost of running the BlockFlow approach in Ethereum gas for N number of FL clients and R number of FL rounds as stated by Mugunthan et al. [12]. With the current Ethereum price of 2,935.36 Euro[2] per Ethereum coin, the blockchain cost of one FL session consisting of ten FL rounds and ten clients would be approximately 1,568.79 Euros. Therefore, assuming one FL session per month is done, it would mean the blockchain cost of 18,825.49 Euros per year.

[2]https://www.coinbase.com/de/price/ethereum. Visited on: 17.01.2022

$$gas(N, R) = 31913 * N + 542045 * R + 477050 * N * R \tag{7.1}$$

On the other hand, deploying a permissioned blockchain such as Hyperledger Fabric would have an initial cost for the hardware needed and the ongoing energy costs. The energy consumption formula of a Raspberry Pi 3 Model B+ proposed by Kesrouani et al. [64] is shown in Equation 7.2, where u is the CPU consumption in percentage.

$$P_{pi} = 3.4495 * u + 3.8563(W) \tag{7.2}$$

We use Equation 7.2 for computing an approximate cost of the approach. The energy consumption of the Raspberry Pi's and the blockchain's idle state is considered. Furthermore, the cost of 0.2664 Euro per Watts per hour [65] is assumed. The CPU usage for the Fabric components in a FL session and in their idle state are computed using the data shown in Figure 7.2. Assuming the same setting as before, the total cost of thriving the ten-client fabric network with ten Raspberry Pi 3 Model B+ would be around 93.68 Euros per year. The calculated yearly blockchain costs of HyperFlow are around 0.50% of the costs of the BlockFlow approach's blockchain costs for the same setting. This reinforces the statement of HyperFlow being a cost-efficient variant of BlockFlow.

7.2. HyperFlow

This section presents the evaluation of the central model's performance in HyperFlow. Using the Fabric Network Generator presented in Chapter 4, a simulation of HyperFlow clients is made. For this experiment, data from the TurboFan data set was split into the clients available on each FL session. Each HyperFlow client contains its data and containerized environment isolated from all other components. All docker containers connect over a docker bridge network, making it possible only to expose the desired ports of each container. This provides a clear picture of the interface and communications occurring in the network.

Since the amount of data available to one client reduces with the number of clients in the session, we hypothesized that the learning rate plays a crucial role in the performance of the central model. More explicitly, a lower learning rate should be used to achieve a better-performing central model with a smaller data set present on each client. We realized FL sessions with three, five, and ten clients trained with learning rates from 10^{-2} to $5 * 10^{-4}$. The session continued until the convergence of the central model. The performance of the central models for each conjugation is shown in Table 7.1. The central model's performance is evaluated over the four test subsets, and the

Table 7.1.: Central models' root mean square error (RMSE) for federated learning sessions containing 3, 5 and 10 clients and different learning rates.

Topology	learning rate	FD001	FD002	FD003	FD004	Mean
3 Clients	0.01	16.93	25.64	19.19	28.32	22.52
	0.005	16.24	26.13	18.99	28.17	22.38
	0.001	15.25	**26.04**	**17.59**	**27.82**	**21.67**
	0.0005	**15.11**	26.29	17.62	27.97	21.75
5 Clients	0.01	16.42	**25.05**	19.85	**26.83**	22.04
	0.005	15.84	25.94	18.74	27.45	21.99
	0.001	**15.49**	25.93	**18.36**	27.65	**21.85**
	0.0005	16.21	27.27	18.83	29.3	22.90
10 Clients	0.01	**15.14**	**26.38**	**18.49**	**28.98**	**22.25**
	0.005	19.22	31.60	20.54	31.76	25.78
	0.001	22.03	31.36	22.15	31.72	26.81
	0.0005	22.66	32.01	22.50	32.15	27.33

mean is presented to provide an easy comparison. The best performing learning rate for sessions containing three and five clients was 10^{-3} and for the session containing ten clients is 10^{-2}. Therefore, further evaluation will be done only with the best-performing learning rate parameters.

Figure 7.4 (a) shows the performance of the central model in the FL session with three clients. A smooth learning curve is shown for all four subsets. The RMSE of subsets FD001 and FD003 have similar performance and smaller RMSE than FD002 and FD004. This shows the higher complexity of subsets FD002 and FD004 by having six operating conditions instead of one as in FD001 and FD003. The convergence of the central model took eight FL rounds.

Figure 7.4 (b) shows higher volatility in the performance curve of the central model for the FL session with five clients. The RMSE of subsets FD002 and FD004 show a steady decrease, while the ones from subsets FD001 and FD003 fluctuate at the initial rounds and decrease after several FL rounds. After ten FL rounds, the central model reaches convergence.

In Figure 7.4 (c), the performance curve of the central model for the FL session with ten clients is shown. Initially, all subsets have similar performance. After the third FL round, the central model decreases its performance on subsets FD002 and FD004 until the ninth round. Meanwhile, the performance on subsets FD001 and FD003 increases. After 11 FL rounds, the central model reaches convergence.

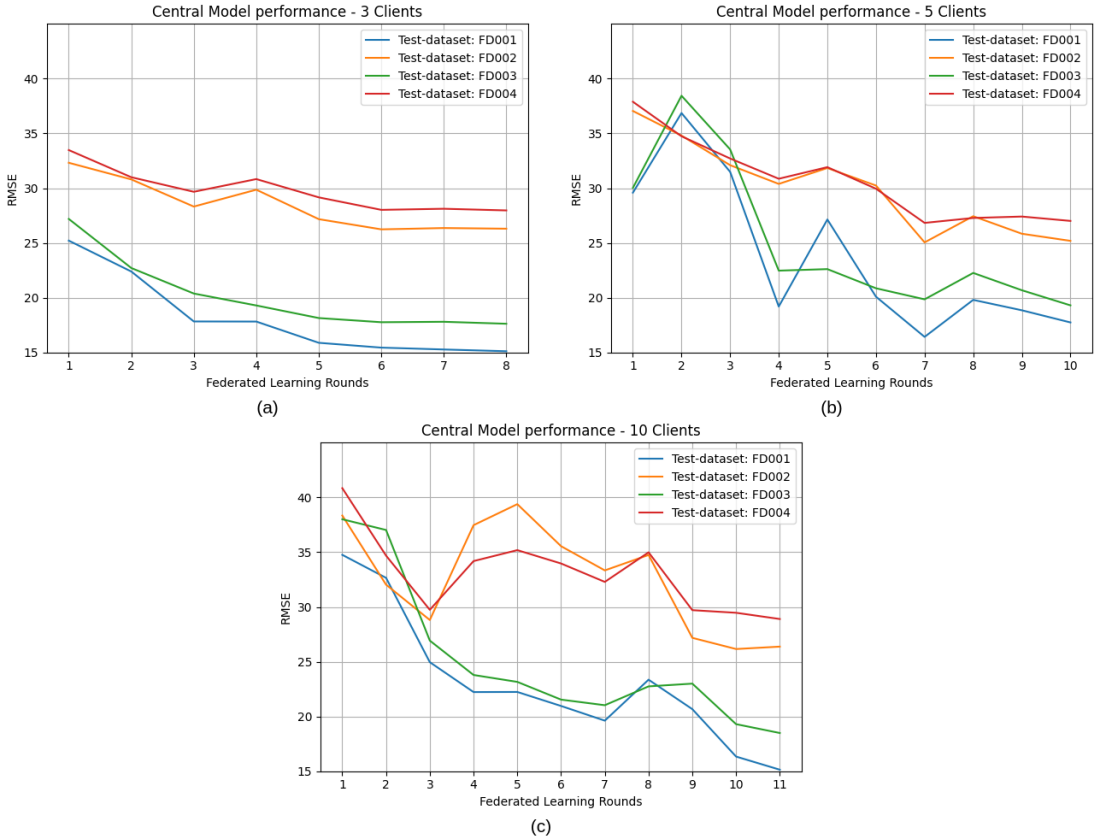

Figure 7.4.: Performance curves of the central model over federated learning rounds for test subsets FD001, FD002, FD003 and FD004. Federated learning session containing (a) 3 clients, (b) 5 clients and (c) 10 clients.

To compare the central model created by different amounts of clients in the FL session, the average RMSE over the four test subsets is calculated and presented for each FL round on Figure 7.5. As the number of clients in an FL session increases, data available to each client decreases. Each local model fits its limited amount of data. Since data disparity increases with the distribution of data over more clients, the trained local models will also diverge more with a higher number of clients present in the FL session. We see this process on the increased fluctuation of the central model's performance over FL rounds with an increase of clients. More FL rounds are required to overcome the higher decentralization of data and provide similar performance of the central model.

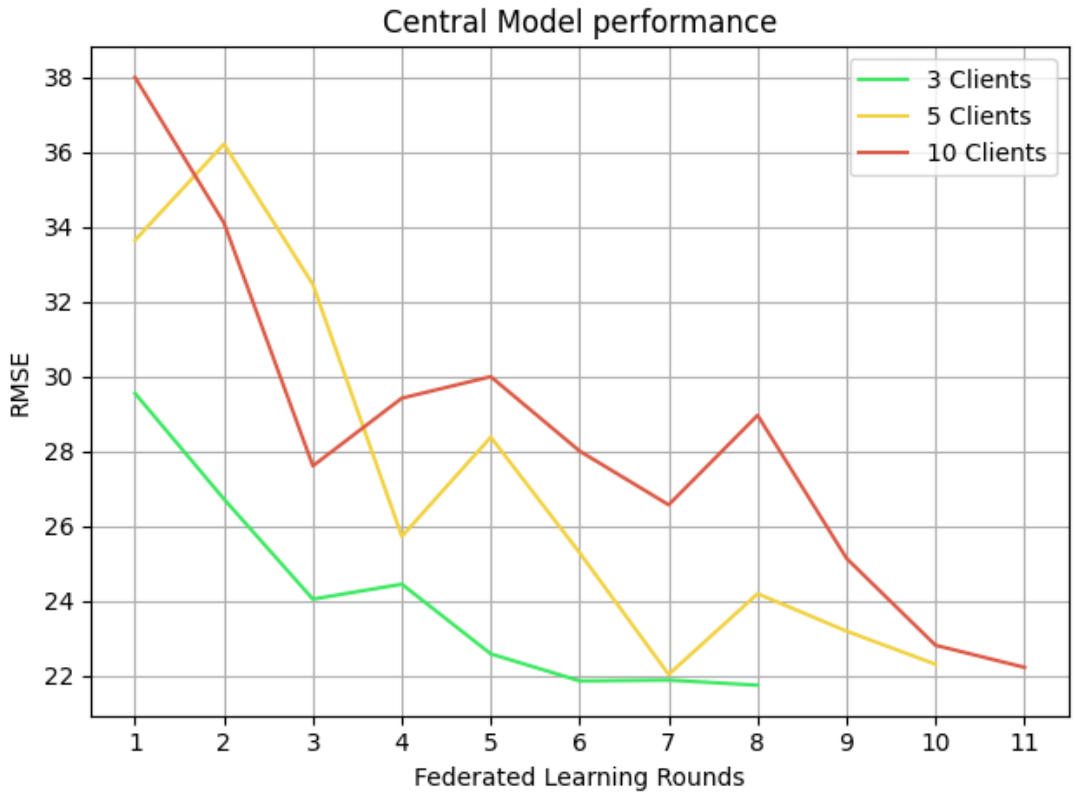

Figure 7.5.: Performance of the central model of 3, 5 and 10 clients averaged over the four test subsets.

7.3. Differential Privacy

The use of DP protects the shared models from GAN reconstruction attacks, gradient leakage attacks, and inference attacks. We evaluate the effect of adding DP to the local models on the central model's performance following the privacy budget proposed by K. Wei et al. [54]. Figure 7.6 shows the effect of privacy budgets of $\epsilon : 100$, $\epsilon : 60$ and $\epsilon : 50$ on the central model's performance over FL rounds for FL sessions with three, five and ten clients. The privacy budget was influenced by the noise added to the trained weights by the Differential Private Keras Adam Optimizer. Models were trained for a total of ten FL rounds.

The performance of the central model with differential privacy for an FL session with three clients is shown in Figure 7.6 (a). The best-performing models are not necessarily the ones of the last FL round. Applying early stopping on the FL session reveals that

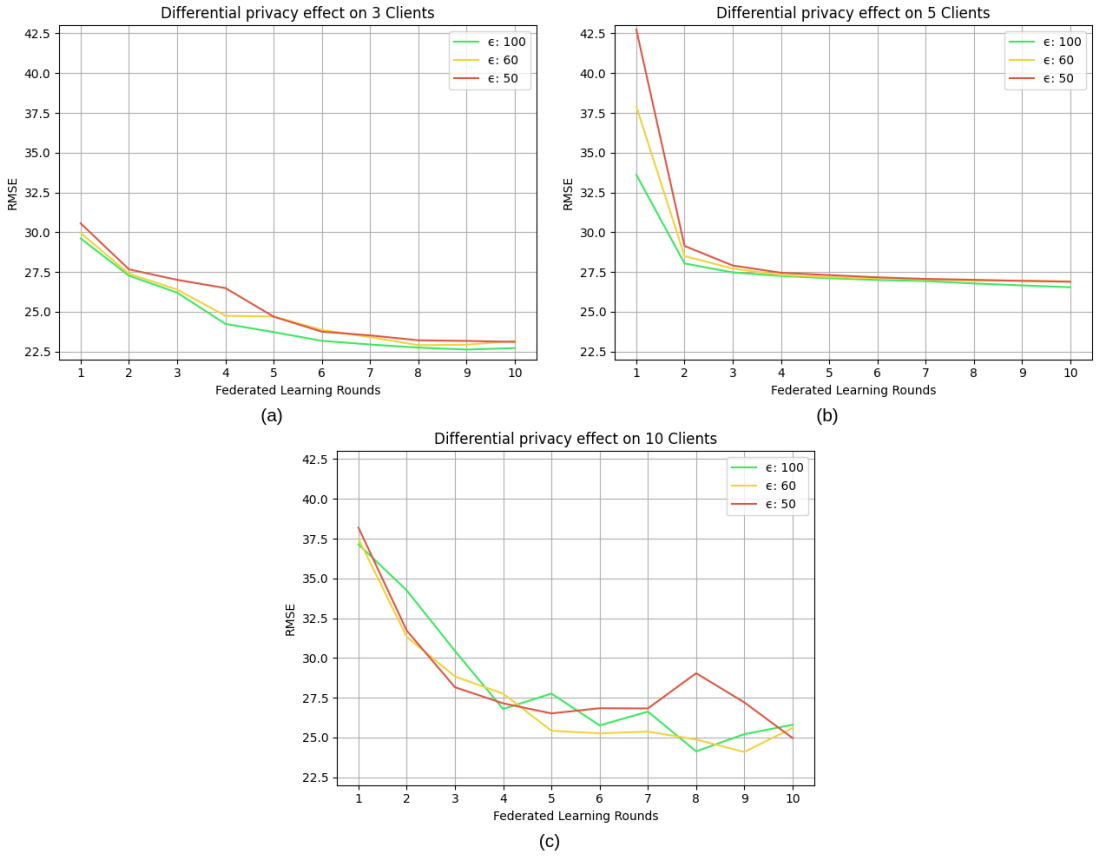

Figure 7.6.: Effect of privacy budgets $\epsilon = 50$, $\epsilon = 60$ and $\epsilon = 100$ on the central model's performance averaged over the four test subsets for a federated learning session containing (a) 3 clients, (b) 5 clients and (c) 10 clients.

The best performing central model for $\epsilon = 100$ is at round nine, $\epsilon = 60$ at round eight, and $\epsilon = 50$ is at round ten. Comparing the results, we see a slight increase in performance for a lower privacy budget. This is in accordance with the expected results since a lower privacy budget means less noise added to the gradients by the optimizer. Increasing the privacy budget from $\epsilon = 50$ to $\epsilon = 60$ reflects an increase of performance of around 0.89%, while the increase of ϵ from $\epsilon = 60$ to $\epsilon = 100$ shows an increase of almost 1.21% in the central model's performance.

The central model's performance for a FL session containing five clients is presented on Figure 7.6 (b). We see that the curves for different privacy budgets are similar in this case. The increase of the privacy budget from $\epsilon = 50$ to $\epsilon = 60$ reflects a decrease of performance of around 0.10%, while the increase of ϵ from $\epsilon = 60$ to $\epsilon = 100$ shows

an increase of around 1.39% in the central model's performance.

In the same way, the central model's performance for an FL session containing ten clients is presented in Figure 7.6 (c). Just as in the FL session without differential privacy, the model presents higher volatility over FL rounds than the ones on FL sessions with a smaller amount of clients. The FL session with an $\epsilon = 100$ presents its peak performance at round eight, the session with $\epsilon = 60$ at round nine, and the one with $\epsilon = 50$ at round ten. The increase of the privacy budget from $\epsilon = 50$ to $\epsilon = 60$ show an increase in performance of almost 3.49%, while the increase of ϵ from $\epsilon = 60$ to $\epsilon = 100$ shows a decrease of around 0.15% in the central model's performance.

7.4. Malicious Clients Threat Analysis

This section describes the analysis done on the robustness of the approach to malicious clients.

Malicious clients are clients providing local models that do not contribute to the increase in performance of the central model over time. Coordinated attacks between several clients are also possible. For the experiment done, malicious clients are aware of other malicious clients and collude together to disrupt the performance of the central model. Each malicious client skips the training process, initializes its local model weights randomly, and uploads it to the cloud. In the evaluation phase, these clients give all other malicious clients a perfect score and the worst score to well-performing clients, called from now on trustful clients.

The robustness of the approach is given by the selection of the aggregation score for a model. From all scores submitted by clients for a model, the median is selected as the aggregation score. The vector is divided by the highest score so that the maximum score equals 1.0. An Apriori for the scores is selected at 0.5, and scores below this Apriori are set to zero. Then, the aggregation score vector is normalized and used to generate the central model. Assuming the majority of clients are trustful clients, using the median ensures the used score is computed by a trustful client.

The experiment was done by instantiating an FL session with ten clients. 40% of those clients are selected as malicious and collude together. To show the importance of the Apriori selected, a lower Apriori of 0.3 was selected for Figure 7.7 and an Apriori of 0.5 was selected for the experiment in Figure 7.8. The robust output would be malicious scores having a nearly zero aggregation score for all FL rounds, while trustful clients are given a relatively high score. In the case of six trustful clients, the optimal scores for trustful clients would be around 0.167.

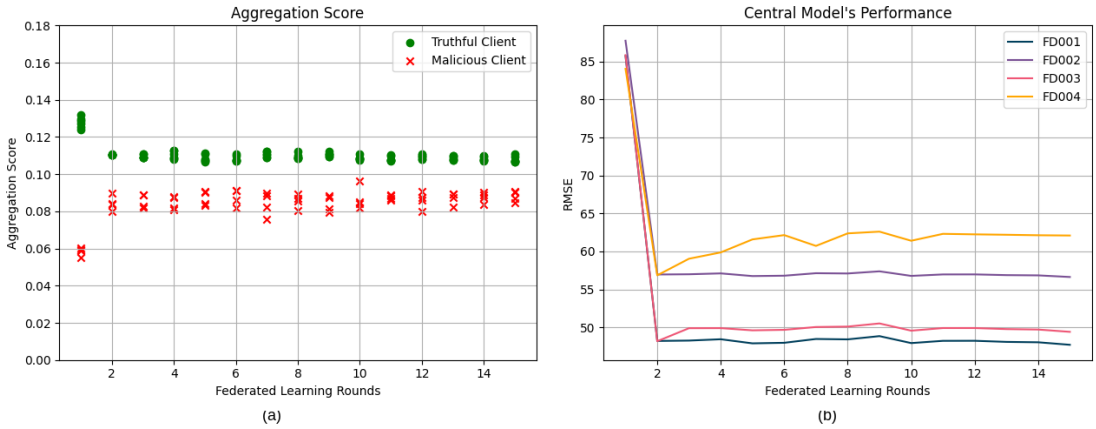

Figure 7.7.: (a) Aggregation scores computed for clients with an Apriori of 0.3. (b) Central model's performance over FL rounds with colluding malicious clients.

Figure 7.7 (a) shows the computed aggregation scores for trustful and malicious clients for an Apriori of 0.3. As seen in the results of Figure 7.7 (a), even malicious models, which are randomly initialized, perform at the baseline of each client's evaluation data set. Additionally, the impact of malicious models increases throughout the FL session. Figure 7.7 (b) shows the central model's performance in the test subsets. Its performance increases when the impact of malicious clients is low, but as the number of rounds increases, the effect of malicious models disrupts the federated learning process.

Setting an Apriori threshold of 0.5 provides the approach with the required robustness. Models having an aggregation score below half the score of the best performing model in the round are not considered for the aggregation. This procedure increases the impact of well-performing models in the aggregation of the central model. Figure 7.8 (a) shows the effects of a well-selected Apriori over the aggregation score of both trustful clients and malicious clients. Scores for clients are as expected. On the one hand, malicious clients obtain a score of zero, so they have no impact on the central model. On the other hand, trustful clients have the maximum impact on the aggregated model. Figure 7.8 (b) presents the central model's performance by adding the Apriori of 0.5 to the procedure. The model is trained until convergence and achieves an average RMSE of 23.19.

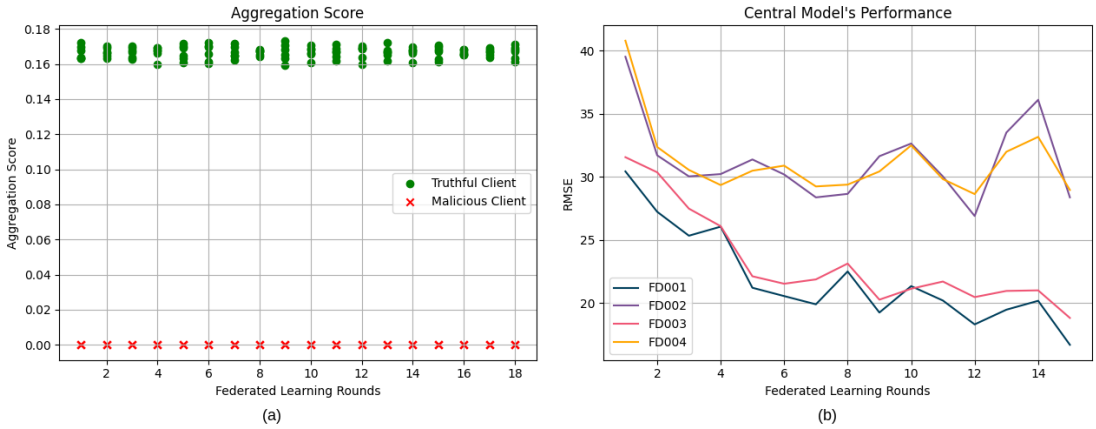

Figure 7.8.: (a) Aggregation scores for clients with Apriori of 0.5. (b) Central model's performance produced with the computed aggregation scores of (a).

7.5. Percentage Deadline Criteria

This section reviews the percentage deadline criteria for the phases of an FL round. The most critical phase is the training phase. This criterion is set to synchronize models advance and be robust against the dropout of clients. The proposed method by Mugunthan et al. [12] is to use hardcoded time deadlines. We propose to use percentage criteria and evaluate its performance.

A realistic environment in predictive maintenance is having specific machines containing higher quality data, e.g., being used in more challenging conditions than others. This is simulated by increasing the complexity of distributed data over the clients with a higher ID number. The data distribution over clients is shown on the left side of Figure 7.9. As presented in Section 6.1, subsets FD001 and FD003 contain data of TurboFan engines on one operating condition. Therefore, data from these subsets are distributed over clients with IDs 1 to 5. On the other hand, data from the subsets FD002 and FD004 is acquired by simulating TurboFan engines in six different operating conditions. This data is distributed over clients with IDs 6 to 10.

On the right side of Figure 7.9, the development of the central model's performance over FL rounds for different percentage criteria as phase deadlines is shown. The sessions are carried out until the convergence of the central model. The session with the 20% percentage criterion trains a central model with data available to clients with ID 01 and 02, which means only data from subset FD001 is used. As the percentage criterion increases to 40%, local models are also trained with partial data of subset FD003. The increase in data quantity does not necessarily mean an increase in the quality of data

Figure 7.9.: (a) Data distribution over the 10 HyperFlow clients. (b) Central model's performance for different stopping percentage criteria.

available. In this case, since subsets FD001 and FD003 have similar characteristics, the central model converges faster but decreases its performance by 1.55%. The increase of the percentage criterion from 40% to 60% lets clients from ID 01 to ID 06 participate in the training of the central model. Here, client 06 provides higher quality data to train a better-performing model. The curve of this percentage criterion shows that the central model takes longer to achieve convergence. Additionally, it seems the data provided by client 06 is not enough to bring the model to increase the performance of the central model produced. The final performance of the central model decreases from the percentage criterion of 40% to 60% by 3.41%. Once the percentage criterion is increased to 80% of clients, the data provided by the clients 07 and 08 increase the central model's performance of 20.05% in comparison with the 60% of clients criterion. Furthermore, the inclusion of all clients provides the best performing model, which presents a performance increase of 10.93% against the 80% of clients criterion.

Table 7.2.: Performance and time per federated learning round for each stopping percentage criterion.

Percentage criterion	Performance [RMSE]	Time per FL round [min]
20% of clients	29.72	3.28
40% of clients	30.18	5.18
60% of clients	31.21	6.43
80% of clients	24.96	8.95
100% of clients	22.23	14.89

Table 7.2 show the performance reached by each percentage criterion and the average time needed for one FL round. The time per FL round is almost entirely needed for the training phase.

8. Conclusion

This chapter concludes this thesis by summarizing the contributions of this work in the integration of a permissioned blockchain to the federated learning process. We briefly state the implementation used in the evaluation of the blockchain together with the methodology used for analyzing key contributions in federated learning. Also, insights provided by the results of this thesis are summarized. Finally, shortcomings of this work and future research paths in this field are specified.

Predictive maintenance has a vast potential for reducing machinery downtime and costs for scheduled maintenance. Federated learning offers the ability to collaborate between organizations to create a well-performing predictive maintenance model. Nevertheless, it has its vulnerabilities, which can be exploited by agents having their own interests in mind. The state-of-the-art approach uses the Ethereum smart contract platform to decentralize the federated learning approach and address federated learning's vulnerabilities such as a compromised server, compromised clients, non-robust aggregation, and gradient leakage vulnerabilities.

We identify shortcomings in the state-of-the-art approach, base our approach on it and address its shortcomings, such as the high blockchain costs and configuration-dependent deadlines in the distributed approach. Additionally, the performance and robustness of the approach were analyzed in a predictive maintenance use case.

We develop HyperFlow, a federated learning framework, which leverages the Hyperledger Fabric blockchain platform as a distributed ledger, which is employed to decentralize the federated learning approach. The costs of HyperFlow are analyzed and compared to the costs of utilizing the state-of-the-art BlockFlow. In addition, we review the performance of HyperFlow using a predictive maintenance publicly available data set to predict the remaining useful lifetime (RUL) of turbofan engines. Furthermore, the robustness of the approach to compromised clients is inspected. Finally, considering the distributed nature of federated learning, we examine the use of percentage stopping criteria as a deadline strategy for submitting local models.

For the implementation, a framework for the automatic configuration and deployment of the Hyperledger Fabric network was developed. Each component is containerized

and communicated over a Docker bridge network. This allows the analysis of the resource consumption of each component. Thereafter, the yearly cost of performing a monthly federated learning session of ten rounds for a total of ten clients is compared between both approaches. Independent of the federated learning use case, the yearly cost of running the HyperFlow approach's blockchain is only about 0.50% of the BlockFlow's blockchain's yearly cost.

HyperFlow was evaluated by the performance of the central model achieved and the robustness of the approach to federated learning vulnerabilities. First, we analyzed Long-Short Term Memory (LSTM) models with different topologies to select a well-performing model for our research. After that, we use this model in HyperFlow to predict the remaining useful lifetime of turbofan engines. The evaluation was done over an increasing amount of clients joining the approach. For it, data was evenly distributed over the available clients. We see the tendency to require a higher number of federated learning rounds to achieve a comparable performance of the central model when increasing the number of clients present.

We validate our approach by generating a central model with data distributed over ten clients, which has comparable performance as a centralized approach. The federated learning central model achieved an average root mean square error (RMSE) of 22.23 on the RUL, while the centralized trained model produced an average RMSE of 19.52.

Additionally, the gradient leakage vulnerability was addressed by adding differential privacy to the models. We analyzed the effects on the central model of having a lower privacy budget on local models, i.e., higher privacy for clients. The changes in performance were minimal since the statistical characteristics of the models are still kept, but higher privacy correlates with a decrease in performance of the central model. We saw no correlation between the performance lost and the number of clients present in the FL round.

We validate the robustness of the approach to the threat of malicious clients. For this experiment, malicious clients collude with one another and provide each other with the best evaluation score and all trustful clients with the worst evaluation score. Using the scoring procedure proposed by Mugunthan et al. [12], we shown the approach is robust to a minority of malicious clients.

Finally, we evaluate the percentage deadline criterion. Data is distributed over ten clients considering a worst-case scenario where valuable data is stored in the clients taking the longest to train their models. An FL session waits until the indicated percentage of clients are ready to continue to the next phase. This is an essential characteristic since clients can drop out of the session due to issues, e.g., connectivity issues, at any time. This deadline criterion provides a trade-off between the time taken for the FL session, robustness against clients dropping out, and the central model's possible performance

gain. In the use case implemented, a percentage deadline of 80% clients would be a good trade-off between the central model's performance, robustness against clients dropping out, and time required for the FL session.

We analyze the shortcomings of the proposed approach. Firstly, for the aggregation of the central model, each client requires access to the local model provided by every other client. The use of a permissioned blockchain provides direct knowledge of the real identity of each client. We address the privacy concerns by implementing differential privacy to the trained models, but statistical characteristics remain. This means, each client would have knowledge of which clients contain high-quality data more frequently. Depending on the use case given, this could not be acceptable. Secondly, the approach does not target the vulnerability of malicious clients performing backdoor attacks. In other words, clients provide a model that performs well on the main task and has a hidden task. On top of that, this leaves a breach for evasion attacks. Lastly, as seen in the results presented in Section 7.2, early stopping in the federated learning process is required to obtain the best performing central model. The selection of the central model was manually made by reviewing the performance of the models in the available test sets.

As concluding remarks, the work in this thesis presents a well-performed implementation and evaluation of a cost-effective federated learning approach that addresses critical vulnerabilities of federated learning. Nevertheless, the mentioned shortcomings should be addressed in future work.

In addition to it, several improvements can be made. An auditing score can be implemented. It is a history score containing information of how far the client has been from the aggregation score selected. If, at some point, a client's auditing score goes below a predefined threshold, an audit about the actions and contributions of the client can be made. In this way, malicious clients would be made accountable for their contributions, and further actions can be taken. Furthermore, the aggregation score calculated can be used for giving financial incentives for clients to contribute high-quality models to the approach. Finally, to perform an automatic early stopping on the rounds taken in a federated learning session, each client could perform an evaluation score with all their data for the central model on each round. After the central model's performance decreases for a given amount of rounds, the session stops, and the best-performing central model is restored.

A. Configuration File

```yaml
hyperflow:
  rounds: 15

bc_connector:
  server_url: "http://server.client_1:3000"

fl_client:
  client_id: 'client_1'
  ca_hostname: "ca.org1.example.com"
  msp_id: 'org1MSP'
  affiliation: 'org1.department2'
  weights_path: '/tmp/fl_client'

dataset:
  base_path: '/tmp/Data/Turbofan_Engine_Dataset/raw'
  dataset_name: 'FD004'
  remaining_sensors: ['s_2', 's_3', 's_4', 's_7', 's_8', 's_9', 's_11', 's_12', 's_13',
                      's_14', 's_15', 's_17', 's_20', 's_21']
  alpha: 0.1

model:
  sequence_length: 30
  nodes_per_layer: [256]
  dropout: 0.1
  activation: 'sigmoid'
  epochs: 30
  batch_size: 128
  use_differential_privacy: True
  l2_norm_clip: 1.0
  noise_multiplier: 0.25
  learning_rate: 0.001
  microbatches: 1
  early_stopping_patience: 5
```

```
34    early_stopping_min_delta: 1
35
36  mlflow:
37    server_url: http://mlflow:5000
38    experiment_name: 'test'
39
40  cloud_client:
41    access_key: '4bdfk1g20bc56ea41ed8'
42    secret_key: 'odUizdenUb3CMcGFoT9LEdOaUrR++mh8V5+na43f'
43    host: 's3-de-central.hostexample.com:80'
44    bucket: 'hyperflow_bucket'
45
46  deadlines:
47    # Modes available: time, percent
48    mode: 'percent'
49    client_percent: 100
```

List of Figures

2.1. Description of the block-chain in Hyperledger Fabric. Source: [35] . . . 10
2.2. Illustration of a transaction's components in Hyperledger Fabric. Source: [35] . . . 11
2.3. Workflow of a submitted transaction in Fabric. Source: [35] . . . 13
2.4. Example of a Fabric network containing four organizations and two channels. Figure was adapted from [35] . . . 14

4.1. Example of the ledger's world-state containing two clients. . . . 27

5.1. Components of a HyperFlow client. . . . 30
5.2. Overview of the phases in a federated learning in HyperFlow. . . . 32
5.3. Workflow of the Client Ready phase. . . . 33
5.4. Workflow of the Training phase. . . . 34
5.5. Workflow of the Validation phase. . . . 35
5.6. Workflow of the Evaluation phase. . . . 36
5.7. Workflow of the Score Decryption phase. . . . 37
5.8. Workflow of the Aggregation phase. . . . 38

7.1. Results on transaction throughput and failed transactions for the analysis done with the Caliper Benchmark Framework. . . . 44
7.2. Average CPU consumption per Fabric component. . . . 45
7.3. Average RAM consumption per Fabric component. . . . 47
7.4. Performance curves of the central model over federated learning rounds for test subsets FD001, FD002, FD003 and FD004. Federated learning session containing (a) 3 clients, (b) 5 clients and (c) 10 clients. . . . 50
7.5. Performance of the central model of 3, 5 and 10 clients averaged over the four test subsets. . . . 51
7.6. Effect of privacy budgets $\epsilon = 50$, $\epsilon = 60$ and $\epsilon = 100$ on the central model's performance averaged over the four test subsets for a federated learning session containing (a) 3 clients, (b) 5 clients and (c) 10 clients. . . 52
7.7. (a) Aggregation scores computed for clients with an Apriori of 0.3. (b) Central model's performance over FL rounds with colluding malicious clients. . . . 54

7.8. (a) Aggregation scores for clients with Apriori of 0.5. (b) Central model's performance produced with the computed aggregation scores of (a). . . 55
7.9. (a) Data distribution over the 10 HyperFlow clients. (b) Central model's performance for different stopping percentage criteria. 56

List of Tables

2.1. Summary of federated learning attacks. Source: [37] 18
2.2. Summary of federated learning defenses. Source: [37] 19

6.1. Summary of the characteristics of TurboFan data set's subsets FD001, FD002, FD003 and FD004. 40
6.2. Summary of parameters used on the optimization of hyper-parameters for the LSTM model. 41
6.3. Results of the hyper-parameter optimization for the LSTM model. . . . 41
6.4. Summary of the best performing LSTM model. 42

7.1. Central models' root mean square error (RMSE) for federated learning sessions containing 3, 5 and 10 clients and different learning rates. . . . 49
7.2. Performance and time per federated learning round for each stopping percentage criterion. 56

Bibliography

[1] M. Pech, J. Vrchota, and J. Bednář. "Predictive Maintenance and Intelligent Sensors in Smart Factory." In: *Sensors* 21.4 (2021), p. 1470.

[2] M. Basingab, L. Rabelo, K. Nagadi, C. Rose, E. Gutiérrez, and W. I. Jung. "Business modeling based on internet of things: a case study of predictive maintenance software using ABS model." In: *Proceedings of the Second International Conference on Internet of things, Data and Cloud Computing*. ICC '17. Association for Computing Machinery, Mar. 2017, pp. 1–5. ISBN: 978-1-4503-4774-7. DOI: 10.1145/3018896.3018905.

[3] S. Roy, J. DeLoach, Y. Li, N. Herndon, D. Caragea, X. Ou, V. P. Ranganath, H. Li, and N. Guevara. "Experimental Study with Real-world Data for Android App Security Analysis using Machine Learning." In: *Proceedings of the 31st Annual Computer Security Applications Conference on - ACSAC 2015*. ACM Press, 2015, pp. 81–90. ISBN: 978-1-4503-3682-6. DOI: 10.1145/2818000.2818038.

[4] J. Goecks, V. Jalili, L. M. Heiser, and J. W. Gray. "How machine learning will transform biomedicine." In: *Cell* 181.1 (2020), pp. 92–101.

[5] S. S. e. Zainab and T. Kechadi. "Sensitive and Private Data Analysis: A Systematic Review." In: *Proceedings of the 3rd International Conference on Future Networks and Distributed Systems*. ICFNDS '19. Association for Computing Machinery, July 2019, pp. 1–11. ISBN: 978-1-4503-7163-6. DOI: 10.1145/3341325.3342002.

[6] H. Zang and J. Bolot. "Anonymization of location data does not work: A large-scale measurement study." In: *Proceedings of the 17th annual international conference on Mobile computing and networking*. 2011, pp. 145–156.

[7] S. Gambs, M.-O. Killijian, and M. Núñez del Prado Cortez. "De-anonymization attack on geolocated data." In: *Journal of Computer and System Sciences*. Special Issue on Theory and Applications in Parallel and Distributed Computing Systems 80.8 (Dec. 2014), pp. 1597–1614. ISSN: 0022-0000. DOI: 10.1016/j.jcss.2014.04.024.

[8] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. "Deep Learning with Differential Privacy." In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. Association for Computing Machinery, Oct. 2016, pp. 308–318. ISBN: 978-1-4503-4139-4. DOI: 10.1145/2976749.2978318.

[9] R. Cramer, I. B. Damgård, et al. *Secure multiparty computation*. Cambridge University Press, 2015.

[10] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava. "A survey on security and privacy of federated learning." In: *Future Generation Computer Systems* 115 (Feb. 2021), pp. 619–640. ISSN: 0167-739X. DOI: 10.1016/j.future.2020.10.007.

[11] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al. "Advances and open problems in federated learning." In: *Foundations and Trends® in Machine Learning* 14.1–2 (2021), pp. 1–210.

[12] V. Mugunthan, R. Rahman, and L. Kagal. "BlockFLow: An Accountable and Privacy-Preserving Solution for Federated Learning." In: *arXiv:2007.03856 [cs, stat]* (July 2020). arXiv: 2007.03856.

[13] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. a. Ranzato, A. Senior, P. Tucker, K. Yang, Q. Le, and A. Ng. "Large Scale Distributed Deep Networks." In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012.

[14] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas. "Communication-Efficient Learning of Deep Networks from Decentralized Data." In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. PMLR, Apr. 2017, pp. 1273–1282.

[15] P. Jiang and L. Ying. "An Optimal Stopping Approach for Iterative Training in Federated Learning." In: *2020 54th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, Mar. 2020, pp. 1–6. ISBN: 978-1-72814-085-8. DOI: 10.1109/CISS48834.2020.1570616094.

[16] A. Saxena and K. Goebel. "Turbofan engine degradation simulation data set." In: *NASA Ames Prognostics Data Repository* (2008), pp. 878–887.

[17] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick. "Hyperledger fabric: a distributed operating

system for permissioned blockchains." In: *Proceedings of the Thirteenth EuroSys Conference*. ACM, Apr. 2018, pp. 1–15. ISBN: 978-1-4503-5584-1. DOI: 10.1145/3190508.3190538.

[18] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund. "Industrial Internet of Things: Challenges, Opportunities, and Directions." In: *IEEE Transactions on Industrial Informatics* 14.11 (Nov. 2018), pp. 4724–4734. ISSN: 1941-0050. DOI: 10.1109/TII.2018.2852491.

[19] R. K. Mobley. *An Introduction to Predictive Maintenance*. Google-Books-ID: SjqXzxpAzSQC. Elsevier, Oct. 2002. ISBN: 978-0-08-047869-2.

[20] G. A. Susto, A. Schirru, S. Pampuri, S. McLoone, and A. Beghi. "Machine Learning for Predictive Maintenance: A Multiple Classifier Approach." In: *IEEE Transactions on Industrial Informatics* 11.3 (June 2015), pp. 812–820. ISSN: 1551-3203, 1941-0050. DOI: 10.1109/TII.2014.2349359.

[21] W. Zhang, D. Yang, and H. Wang. "Data-Driven Methods for Predictive Maintenance of Industrial Equipment: A Survey." In: *IEEE Systems Journal* 13.3 (Sept. 2019), pp. 2213–2227. ISSN: 1937-9234. DOI: 10.1109/JSYST.2019.2905565.

[22] O. Serradilla, E. Zugasti, J. Rodriguez, and U. Zurutuza. "Deep learning models for predictive maintenance: a survey, comparison, challenges and prospects." In: *Applied Intelligence* (2022), pp. 1–31.

[23] A. Graves. "Long Short-Term Memory." In: *Supervised Sequence Labelling with Recurrent Neural Networks*. Ed. by A. Graves. Studies in Computational Intelligence. Springer, 2012, pp. 37–45. ISBN: 978-3-642-24797-2. DOI: 10.1007/978-3-642-24797-2_4.

[24] S. Nakamoto. "Bitcoin: A peer-to-peer electronic cash system." In: *Decentralized Business Review* (2008), p. 21260.

[25] M. Kõlvart, M. Poola, and A. Rull. "Smart Contracts." In: *The Future of Law and eTechnologies*. Ed. by T. Kerikmäe and A. Rull. Springer International Publishing, 2016, pp. 133–147. ISBN: 978-3-319-26896-5. DOI: 10.1007/978-3-319-26896-5_7.

[26] T. H. Meitinger. "Smart Contracts." In: *Informatik-Spektrum* 40.4 (Aug. 2017), pp. 371–375. ISSN: 1432-122X. DOI: 10.1007/s00287-017-1045-2.

[27] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang. "An overview of blockchain technology: Architecture, consensus, and future trends." In: *2017 IEEE international congress on big data (BigData congress)*. IEEE. 2017, pp. 557–564.

[28] D. Yaga, P. Mell, N. Roby, and K. Scarfone. *Blockchain technology overview*. NIST IR 8202. Oct. 2018, NIST IR 8202. DOI: 10.6028/NIST.IR.8202.

[29] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou. "A survey of distributed consensus protocols for blockchain networks." In: *IEEE Communications Surveys & Tutorials* 22.2 (2020), pp. 1432–1465.

[30] K. Wu, M. Liu, and J. Xu. "How Will Blockchain Technology Impact Auditing and Accounting: Permissionless Vs. Permissioned Blockchain." In: *Current Issues in Auditing* 13 (Aug. 2019). DOI: 10.2308/ciia-52540.

[31] J. Polge, J. Robert, and Y. Le Traon. "Permissioned blockchain frameworks in the industry: A comparison." In: *ICT Express* 7.2 (June 2021), pp. 229–233. ISSN: 2405-9595. DOI: 10.1016/j.icte.2020.09.002.

[32] E. Zhou, H. Sun, B. Pi, J. Sun, K. Yamashita, and Y. Nomura. "Ledgerdata refiner: a powerful ledger data query platform for hyperledger fabric." In: *2019 sixth international conference on Internet of Things: Systems, Management and Security (IOTSMS)*. IEEE. 2019, pp. 433–440.

[33] M. Rauchs, A. Blandin, K. Bear, and S. B. McKeon. *2nd Global Enterprise Blockchain Benchmarking Study*. ID 3461765. Sept. 2019. DOI: 10.2139/ssrn.3461765.

[34] J. A. Chacko, R. Mayer, and H.-A. Jacobsen. "Why do my blockchain transactions fail? a study of hyperledger fabric." In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, pp. 221–234.

[35] T. L. Foundation. *Hyperledger Fabric: A blockchain platform for the enterprise*. URL: https://hyperledger-fabric.readthedocs.io/en/release-2.2/ (visited on 12/15/2021).

[36] T. Liwei and S. Yu. "Research Summary of Blockchain Fragmentation Propagation Mechanism Based on Merkel Tree." In: *Journal of Physics: Conference Series* 1914.1 (May 2021), p. 012010. ISSN: 1742-6596. DOI: 10.1088/1742-6596/1914/1/012010.

[37] N. Bouacida and P. Mohapatra. "Vulnerabilities in Federated Learning." In: *IEEE Access* 9 (2021), pp. 63229–63249. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3075203.

[38] H. Bae, J. Jang, D. Jung, H. Jang, H. Ha, H. Lee, and S. Yoon. "Security and Privacy Issues in Deep Learning." In: *arXiv:1807.11655 [cs, stat]* (Mar. 2021). arXiv: 1807.11655.

[39] G. Sadowsky, J. X. Dempsey, A. Greenberg, B. J. Mack, and A. Schwartz. *Information Technology Security Handbook*. World Bank, 2013. ISBN: 978-0-9747888-0-7.

[40] C. Ma, J. Li, M. Ding, H. H. Yang, F. Shu, T. Q. S. Quek, and H. V. Poor. "On Safeguarding Privacy and Security in the Framework of Federated Learning." In: *IEEE Network* 34.4 (July 2020), pp. 242–248. ISSN: 1558-156X. DOI: 10.1109/MNET.001.1900506.

[41] W. Wei, L. Liu, Y. Wut, G. Su, and A. Iyengar. "Gradient-Leakage Resilient Federated Learning." In: *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. July 2021, pp. 797–807. DOI: 10.1109/ICDCS51616.2021.00081.

[42] Y. Lu and L. Fan. "An Efficient and Robust Aggregation Algorithm for Learning Federated CNN." In: *Proceedings of the 2020 3rd International Conference on Signal Processing and Machine Learning*. SPML 2020. Association for Computing Machinery, Oct. 2020, pp. 1–7. ISBN: 978-1-4503-7573-3. DOI: 10.1145/3432291.3432303.

[43] S. Li, Y. Cheng, Y. Liu, W. Wang, and T. Chen. "Abnormal Client Behavior Detection in Federated Learning." In: (Oct. 2019).

[44] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer. "Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent." In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017.

[45] S. Shen, S. Tople, and P. Saxena. "A uror: defending against poisoning attacks in collaborative deep learning systems." In: *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACM, Dec. 2016, pp. 508–519. ISBN: 978-1-4503-4771-6. DOI: 10.1145/2991079.2991125.

[46] F. Ang, L. Chen, N. Zhao, Y. Chen, W. Wang, and F. R. Yu. "Robust Federated Learning With Noisy Communication." In: *IEEE Transactions on Communications* 68.6 (June 2020), pp. 3452–3464. ISSN: 1558-0857. DOI: 10.1109/TCOMM.2020.2979149.

[47] Y. Qu, L. Gao, T. H. Luan, Y. Xiang, S. Yu, B. Li, and G. Zheng. "Decentralized Privacy Using Blockchain-Enabled Federated Learning in Fog Computing." In: *IEEE Internet of Things Journal* 7.6 (June 2020), pp. 5171–5183. ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.2977383.

[48] L. Peng, W. Feng, Z. Yan, Y. Li, X. Zhou, and S. Shimizu. "Privacy preservation in permissionless blockchain: A survey." In: *Digital Communications and Networks* 7.3 (Aug. 2021), pp. 295–307. ISSN: 2352-8648. DOI: 10.1016/j.dcan.2020.05.008.

[49] J. Sun, Y. Wu, S. Wang, Y. Fu, and X. Chang. "A Permissioned Blockchain Frame for Secure Federated Learning." In: *IEEE Communications Letters* (2021), pp. 1–1. ISSN: 1558-2558. DOI: 10.1109/LCOMM.2021.3121297.

[50] Y. Li, C. Chen, N. Liu, H. Huang, Z. Zheng, and Q. Yan. "A Blockchain-based Decentralized Federated Learning Framework with Committee Consensus." In: *IEEE Network* 35.1 (Jan. 2021). arXiv: 2004.00773, pp. 234–241. ISSN: 0890-8044, 1558-156X. DOI: 10.1109/MNET.011.2000263.

[51] H. B. Desai, M. S. Ozdayi, and M. Kantarcioglu. "BlockFLA: Accountable Federated Learning via Hybrid Blockchain Architecture." In: *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy*. CODASPY '21. Association for Computing Machinery, Apr. 2021, pp. 101–112. ISBN: 978-1-4503-8143-7. DOI: 10.1145/3422337.3447837.

[52] M. A. Rahman, M. S. Hossain, M. S. Islam, N. A. Alrajeh, and G. Muhammad. "Secure and Provenance Enhanced Internet of Health Things Framework: A Blockchain Managed Federated Learning Approach." In: *IEEE Access* 8 (2020), pp. 205071–205087. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3037474.

[53] Y. Lu, X. Huang, Y. Dai, S. Maharjan, and Y. Zhang. "Blockchain and Federated Learning for Privacy-Preserved Data Sharing in Industrial IoT." In: *IEEE Transactions on Industrial Informatics* 16.6 (June 2020), pp. 4177–4186. ISSN: 1941-0050. DOI: 10.1109/TII.2019.2942190.

[54] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. Quek, and H. V. Poor. "Federated learning with differential privacy: Algorithms and performance analysis." In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 3454–3469.

[55] W. Zhang, Q. Lu, Q. Yu, Z. Li, Y. Liu, S. K. Lo, S. Chen, X. Xu, and L. Zhu. "Blockchain-Based Federated Learning for Device Failure Detection in Industrial IoT." In: *IEEE Internet of Things Journal* 8.7 (Apr. 2021), pp. 5926–5937. ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.3032544.

[56] H. Kim, J. Park, M. Bennis, and S.-L. Kim. "Blockchained On-Device Federated Learning." In: *IEEE Communications Letters* 24.6 (June 2020), pp. 1279–1283. ISSN: 1558-2558. DOI: 10.1109/LCOMM.2019.2921755.

[57] S. Otoum, I. Al Ridhawi, and H. T. Mouftah. "Blockchain-Supported Federated Learning for Trustworthy Vehicular Networks." In: *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*. Dec. 2020, pp. 1–6. DOI: 10.1109/GLOBECOM42002.2020.9322159.

[58] G. Duarte Pasa, I. Paixão de Medeiros, and T. Yoneyama. "Operating Condition-Invariant Neural Network-based Prognostics Methods applied on Turbofan Aircraft Engines." In: *Annual Conference of the PHM Society* 11.1 (Sept. 2019). ISSN: 2325-0178, 2325-0178. DOI: 10.36001/phmconf.2019.v11i1.786.

[59] K. T. Chui, B. B. Gupta, and P. Vasant. "A Genetic Algorithm Optimized RNN-LSTM Model for Remaining Useful Life Prediction of Turbofan Engine." In: *Electronics* 10.33 (Jan. 2021), p. 285. DOI: 10.3390/electronics10030285.

[60] Y. Wu, M. Yuan, S. Dong, L. Lin, and Y. Liu. "Remaining useful life estimation of engineered systems using vanilla LSTM neural networks." In: *Neurocomputing* 275 (Jan. 2018), pp. 167–179. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2017.05.063.

[61] S. Zheng, K. Ristovski, A. Farahat, and C. Gupta. "Long Short-Term Memory Network for Remaining Useful Life estimation." In: *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*. June 2017, pp. 88–95. DOI: 10.1109/ICPHM.2017.7998311.

[62] A. Listou Ellefsen, E. Bjørlykhaug, V. Æsøy, S. Ushakov, and H. Zhang. "Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture." In: *Reliability Engineering & System Safety* 183 (Mar. 2019), pp. 240–251. ISSN: 0951-8320. DOI: 10.1016/j.ress.2018.11.027.

[63] H. V. Düdükçü, M. Taşkıran, and N. Kahraman. "LSTM and WaveNet Implementation for Predictive Maintenance of Turbofan Engines." In: *2020 IEEE 20th International Symposium on Computational Intelligence and Informatics (CINTI)*. Nov. 2020, pp. 000151–000156. DOI: 10.1109/CINTI51262.2020.9305820.

[64] K. Kesrouani, H. Kanso, and A. Noureddine. "A Preliminary Study of the Energy Impact of Software in Raspberry Pi devices." In: *2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. Sept. 2020, pp. 231–234. DOI: 10.1109/WETICE49692.2020.00052.

[65] Statista. *Industrial electricity prices including tax Germany 1998-2022*. Mar. 2022. URL: https://www.statista.com/statistics/1050448/industrial-electricity-prices-including-tax-germany/ (visited on 03/09/2022).