Towards Blockchain Network Platform for IoT Data Integrity and Scalability

by

Chung Yup Kim

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Engineering
in Network Engineering.

Victoria University of Wellington
2020

Abstract

Decentralised technology backed by blockchain has gained popularity in recent years, as it secures autonomous ecosystems without the need for a central authority. The blockchain concept originated in the financial domain using cryptocurrency but has been applied to a variety of industries over the last few years. In the era of Industry 4.0, most enterprises leverage automation by using Internet of Things (IoT) technology. Despite the numerous applications of blockchain across industries, significant latency in the consensus algorithm in blockchain hinders its adoption among businesses using IoT technology. A number of studies have addressed the obstacles of transaction processing performance and system scalability, mostly based on a public blockchain. However, the approaches still involve centralised components and thus fail to fully utilise decentralisation. Here, a private blockchain-based IoT data integration platform is proposed to achieve data integrity and system scalability. Along with a lightweight IoT gateway, instead of any other additional middleware, the process and the system configuration are streamlined. By using Hyperledger Fabric, the design is validated, and the proposed architecture outperforms other conventional models in IoT data processing. Thus, decentralisation in IoT environments is achieved.

Acknowledgments

I dedicate this achievement to my mother in heaven who always encouraged me. I thank my wife Hyeonhwa, and my fur babies Daisy and Lily for their patience during my absence.

Foremost, I give sincere gratitude to my supervisors, Professor Winston Seah, who has supported all of my research from the beginning with precious advice, and Dr. Bryan Ng, who has given valuable and practical advice in a technical aspect.

I would wish to express special appreciation to Dr. Alvin Valera and Dr. Diana Siwiak, who have advised me on the overall research work at every important stage.

I would like to extend many thanks to my peers in WiNe lab. Their ways of research approach provided with a fairly clear stimulus and had a profound effect on my research.

The work in this thesis was supported by the scholarship, "Victoria Huawei NZ Research Programme." The sponsorship is also highly appreciated.

Contents

Chapter 1

Introduction

In the era of Industry 4.0, automation prevails significantly in manufacturing. In particular, big data, Internet of Things (IoT), blockchain, and artificial intelligence (AI) technologies in the smart factory garner much attention.

With regard to IoT environments, data volume, fault tolerance, and generation frequency are key considerations. Full automation can only be achieved when each process is seamlessly integrated and any single component does not affect service downtime.

Blockchain is appropriate for process automation along with high availability (HA). However, because of certain drawbacks, such as significant latency in transaction processing, the application of blockchain to IoT environments lags behind its use in other domains. A number of groups have been working towards performance improvements, but their approaches do not fully utilise blockchain as they include additional centralised components.

Therefore, this thesis aims at developing a blockchain network platform to integrate IoT data effectively. Among a variety of IoT environments, facility management in manufacturing based on IoT sensor data is selected as a model. To facilitate application of the platform to generic IoT environments, scalability and flexibility in system configuration are taken into account.

1.1 Research Motivation

Various forms of process automation in manufacturing are accelerated by IoT technology. The core of this technology is the processing of a large volume of data gathered from IoT sensors. As it is the foundation for the autonomous operation of production equipment, data integrity is paramount in IoT.

IoT data are ubiquitous in diverse environments. Communication is established between instruments ranging from small devices to back-end legacy systems in manufacturing. Cloud computing and distributed computing architectures are most commonly used for better performance and efficiency with regard to IoT data [1].

Both architectures are based on centralisation and have inherent drawbacks such as security issues and a single point of failure (SPOF). To overcome such shortcomings, the blockchain [2], a decentralised architecture, has been introduced. Decentralisation is based on autonomous computing without any authority or intermediary. Accordingly, a blockchain-based system is free from a SPOF and agile in operational decisions. It is also more robust than distributed computing with a central node, as all the processing data are shared with each node.

Despite its strengths, blockchain has weaknesses in performance and scalability, mainly caused by the consensus algorithm. These weaknesses hinder blockchain adoption in IoT environments, as large volumes of data are generated continuously by IoT devices. However, not all IoT data are necessarily meaningful. IoT sensors can seamlessly detect anomalies in the environment, and only those occasional anomalies need further attention. Selective data processing on time is critical. In this regard, blockchain-based IoT data processing needs to be tailored while guaranteeing data integrity [3].

Many studies have focused on solving these problems to promote blockchain adoption in IoT environments [4]. Although these approaches are effective, they employ additional centralised components, imposing a potential SPOF. Therefore, this thesis intends to tackle the challenges facing blockchain adoption in IoT environments and to propose a new data integration model to ensure flexibility and reconfigurability with full decentralisation.

1.2 Research Problem

For operation automation in manufacturing, data generated by IoT sensors are sent to the actuators of the facility management system, which are switched to be enabled or disabled based on the data they receive.

The integrity of the sensed data must be preserved to ensure accurate control. So that the actuators' action history can be tracked, the data must be immutable and traceable. In addition, the architecture must accommodate endlessly generated data without performance degradation.

Based on these requirements, the problem statement is defined as follows:

Problem Statement

Given that control of actuators for smart factory production lines in manufacturing involves:

1. IoT-based sensor data acquisition, and
2. adjustment of actuators in accordance with these data,

determine a blockchain-based IoT data integration platform model, while:

1. ensuring data reliability, immutability, and traceability;
2. minimising latency in data processing; and

3. streamlining data processing,

subject to constraints in IoT environments on:

1. computational hardware resources,
2. heterogeneous components, and
3. sensor networks.

1.3 Research Objectives

This research aims to propose a blockchain-based network platform for an IoT-based facility management system in the smart factory with the following objectives:

1. reinforce data integrity and guarantee seamless processing under IoT-specific environmental constraints,
2. assure high transaction processing performance,
3. secure the interface in between IoT devices and blockchain network, and
4. configure the system without any additional centralised system components.

1.4 Research Contributions

The ultimate goal of this research is to explore an effective data processing platform in IoT environments that can be utilised in many potential business areas. The thesis addresses technologies to build the foundation for this purpose. As a base for the big picture, the methods to guarantee IoT data integrity and system scalability are discussed along with the following aspects:

- For the operation automation using IoT in manufacturing, to which blockchain application is regarded not to be the best solution, a private blockchain network model is presented. A private model secures data privacy for an organisation, and mitigate performance degradation caused by an anonymous consensus in conventional blockchain.

- To overcome the downside of blockchain, an enhanced system configuration is introduced while decentralisation is fully achieved. All the processing data are stored in distributed file systems and only anomalous data are processed further. Without any additional centralised components, the system is fault-tolerant.

- The proposed system architecture constitutes the core part for the tokenized economy that integrates autonomous supply chain management (SCM) system for hardware parts. Blockchain has been originally developed for cryptocurrency, and the token implementation on the blockchain enables the integration.

1.5 Outline of Thesis

This thesis is composed of six chapters.

Chapter 1. Introduction

The problems facing blockchain adoption in IoT environments are discussed, and the aim of the research is presented.

Chapter 2. Background and Related Work

General blockchain features are described, and the limitations of the conventional blockchain architecture are analysed. Then, recent studies on the existing issues are reviewed, and their advantages and disadvantages are compared.

Chapter 3. Blockchain-based Network Design

The design of the proposed architecture to overcome the restrictions is presented. First, factors to be considered in the design approach are enumerated. Then, conceptual design and application scenarios are described. Lastly, the proposed architecture is presented with a model system design.

Chapter 4. Validation

Validation of the proposed architecture is discussed, and a test system is elaborated in more detail for the evaluation of the IoT data integration.

Chapter 5. Experimental Results

Functional and non-functional test methods are highlighted, and according to each test scenario, experimental results are shown and analysed. In addition, data interface methods are investigated further.

Chapter 6. Conclusion and Future Work

The thesis is concluded and future work for continuous improvements is suggested.

Chapter 2

Background and Related Work

This chapter explores IoT and blockchain technologies. The weaknesses of blockchain and limitations to its adoption, especially with regard to IoT-specific features, are introduced.

First, major characteristics of both IoT and blockchain are presented in Sections 2.1 and 2.2, respectively. Since this thesis focuses on blockchain application to IoT environments rather than the technological aspects of IoT, more in-depth studies of blockchain are discussed.

Next, Section 2.3 depicts general blockchain classification and Section 2.4 deals with the necessities of blockchain adoption. These two sections establish which type of blockchain is needed is various scenarios and why.

Then, in Section 2.5, the limitations of conventional blockchains in relation to IoT environments are covered in a practical way.

Lastly, existing related works are reviewed and compared with the main approach of this research.

2.1 Characteristics of IoT Networks

The IoT network is an interaction system in which communication is performed machine-to-machine (M2M) without human interventions. In this section, IoT characteristics are discussed with regard to the manufacturing context.

Constrained Resources

The main difference between IoT networks and traditional networks is the constrained resources of IoT devices. IoT devices typically have low memory capacity, low computational power, and low battery life and hence lack the hardware resources necessary to ensure security, making them vulnerable to external attacks. To defend against various attacks, traditional networks have secure infrastructure such as firewalls, intrusion prevention systems (IPS) and intrusion detection systems (IDS), all of which are host-based and cannot be applied to resource-constrained IoT devices.

Lack of Standardisation and Compatibility

In addition to hardware resource constraints, IoT devices are lacking in software updates, external access control, and other self-protection mechanisms. Furthermore, the heterogeneity of devices produced by a variety of manufacturers presents challenges in terms of standardisation and compatibility.

For these reasons, IoT environments entail additional system components: middleware, brokers, and gateways. IoT gateways based on an open-source hardware have been highlighted in recent years [5][6]. These gateways are so versatile and lightweight that they can be utilised in various cases with low costs.

Large Data Volume

IoT devices produce a huge amount of data. Moreover, with the advent of 5G networks [7], transmission will be accelerated, data packet size will be increased, and computational nodes will facilitate various manipulations of the data.

Therefore, optimising the configuration of the whole system to accommodate the extended capacity towards 5G networks without any hot spots will be significant. All the IoT components should be harmonised and streamlined according to the business requirements.

Frequent Transaction Occurrences

Compared with human interactions, IoT networks have much more frequent transactions because a number of devices generate data continuously. Once IoT devices have been deployed, they produce data both periodically and whenever certain events occur. Mostly the packet size is small and the transmission is one way from devices to legacy systems.

Operational Transaction Usage

The payloads can be explicitly defined depending on business requirements. Most transactions intend to trigger relevant actions rather than to represent a decisive status. For example, in financial transactions, each transaction is likely to declare a certain change of the current situation to be recorded. By contrast, IoT sensor data are used to control other connected devices.

2.2 Characteristics of Blockchain

This section provides an overview of blockchain technology, well-known common attributes, and collaborative features with other technologies.

2.2.1 Introduction to Blockchain

Blockchain Overview

Blockchain is not a new invention but rather embraces and orchestrates existing technologies such as distributed ledger technology (DLT), decen-

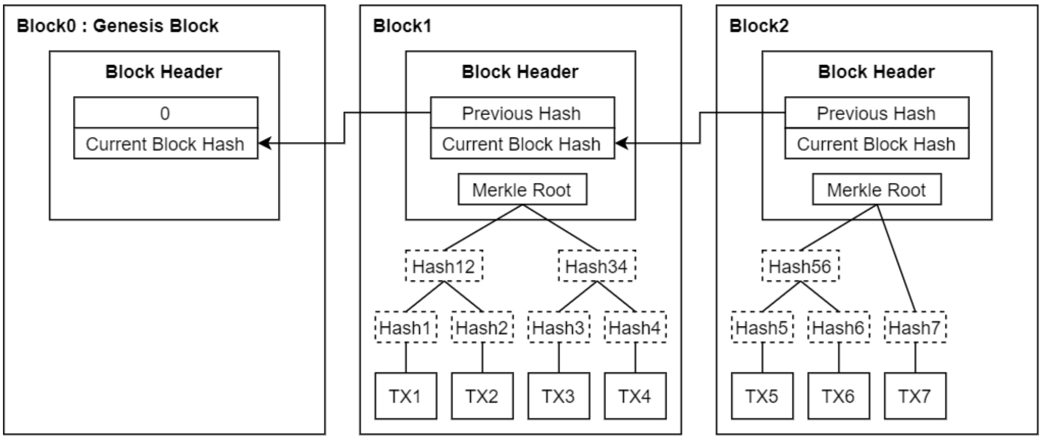

Figure 2.1: Blockchain structure

tralisation, peer-to-peer (P2P) technology, cryptography, hash functions, RESTful API [8], and digital consensus algorithms. It explicitly includes those technologies so as to connect all the participants and to process transactions in the network based on the decentralised architecture.

Decentralised Technology

Decentralisation means that no one rules over or controls everything in the whole system, and there is no SPOF. Once the need for trusted third parties or intermediaries is removed, efficiency, scalability, and potential values emerge due to the independence of processes. In addition, distributing the computation to each node in the network increases sustainability and efficient resource utilisation [1]. This idea of autonomous decentralisation is entirely underpinned by blockchain technology.

Structure of Blockchain

Blockchain is a cryptographically secured chain of blocks based on anonymous consensus using a distributed P2P network.

As shown in Figure 2.1, a block consists of a series of transactions. Every transaction is published to its own hash value, and then the hash values are aggregated to a hash again in a block. The block header includes this aggregated hash, current block hash, and the previous block's hash. Thus, all the blocks in a blockchain are tightly coupled, so that modification of any block requires changing all the blocks, which is impossible. In this way, data consistency in a blockchain is guaranteed.

The first block is called a genesis block and is generated by the creator of the blockchain. It has a zero hash value for the previous block's hash. After the creation of the blockchain, new blocks that reach consensus among participants are added to the existing blockchain serially.

History of Blockchain

The blockchain concept was introduced by a person with the pseudonym Satoshi Nakamoto [2]. Blockchain was initially aimed at solving problems, such as 'double spending' and 'Byzantine Fault Tolerance', underlying the cryptocurrency Bitcoin. Since then, it has been applied to various altcoins, or alternative cryptocurrencies.

The architecture of the consensus-based verification and the shared ledger was the focus for cryptocurrency or contract-based interactive businesses. Subsequently, for further applications, Ethereum emerged as a 'turing complete' blockchain platform powered by smart contract technology [9]. Since then, other blockchain platforms that are analogous to Ethereum or structured with different consensus algorithms have been developed [10, 11, 12].

Smart Contract

The smart contract is closely associated with blockchain. In fact, it has transformed blockchain innovatively from the level of a simple distributed ledger to a platform for numerous application services with the broad concept of decentralised applications (DApps) [13].

The smart contract was introduced with Ethereum but is commonly used to refer to a self-executing contract on most blockchains. It is a computer program code that can be written in various programming languages. The rules of transaction processing in relation to a business are implemented in the smart contract.

The smart contract is based on a trigger for further processing that is set on or off when a certain requirement is met. The smart contract literally infers the adherence of an agreement in some cases by defining the regulations in the program logic. In addition, it facilitates the automation of the entire process. All the operational processing logic is aggregated in the smart contract, and each decisive procedure can be dealt with by the predefined rules.

2.2.2 Attributes of Blockchain

The most representative features of blockchain are as follows: transparency, immutability, traceability, and autonomy [14, 15, 16].

Transparency

All the transactions and data are open to each participant on the blockchain: all information except for a participant's identity is shared. The transaction or data owner remains anonymous. This sharing creates transparency that brings about trust between trustless entities.

Some types of blockchain have data privacy capabilities to limit participants' access to certain information by using private network channels. These network channels use a subnet or different layer of the blockchain network. All participants join the blockchain network, but only permitted members can access a specific channel. The channel becomes exclusive for those who need to keep data private from others. In this case, transparency is still guaranteed within the channel.

Immutability

As shown in Figure 2.1, all the blocks on the blockchain are connected sequentially. Accordingly, once the blocks are connected after validation, the contents in each block are nearly impossible to modify or delete. This is because when a new block is created, the hash of the previous block is included in the new block. In order to modify or delete transactions in a block, or even a block itself, all the blocks must be changed at once.

One can imagine that this could be accomplished by using powerful computational hardware, but during the modification, a new block is added again and again. As a result, the modification becomes impossible, and immutability is achieved.

Therefore, data on blockchain are secure.

Traceability

As data on blockchain are immutable and all blocks are connected, any data can be tracked down. All the transactions are serialised on blockchain, so that any transaction can be easily traced by following the hash of the previous block.

This feature is particularly useful in audit and SCM systems. If a transaction can be traced, it becomes easier to identify the cause of problems. In SCM, particularly, traceability can find the provenance of products and reinforce the visibility of distribution. This engenders trust in customers and eventually increases sales.

Autonomy

Blockchain, fundamentally, aims for autonomy without any central authority. This can be translated into availability and autonomous processing.

Since blockchain is deployed with a distributed shared ledger on all the participating nodes, data can be preserved in some nodes, regardless of other nodes' state. Consequently, when a node crashes, the system continues to operate without service downtime and is never affected by the crashed node. Blockchain can stand independent of individual nodes.

For many system operations, autonomous processing is necessary because manual activities are error-prone and difficult to maintain within a complex enterprise application system.

Autonomy by blockchain is distinguishable from automated operations. Database procedure scripts and batch processing can be used for automation, but they can only manage simple events in each programmable step using binary decisions. Such automated procedures dealing with simple events are inadequate for intricate circumstances. When a more complex decision must be made based on the result of a predecessor, the need for self-regulation increases. Blockchain accommodates autonomy by implementing smart contracts. Moreover, it is applicable system-wide.

As presented in the introduction to the smart contract in the previous section, the smart contract includes a series of all the processing logic. All the operations are governed by the predefined logic, which represents a pipeline of a whole process lifecycle. Every step in which a decision is based on the preceding result goes through validation from the majority of nodes or additional check logic. The smart contract controls all the data processing and is commonly used by all the nodes. Thus, autonomy is achieved.

2.3 Blockchain Taxonomy

Blockchains are classified based on how they control participation, how they control data, and what participants can do. Public (or permissionless) blockchains can be differentiated from private (or permissioned) ones. Public or permissionless blockchains are regarded as the same thing, as are private or permissioned blockchains, in the case of networks mainly used for cryptocurrencies.

However, in the case of IoT, they should be distinguished. Authentication is the criterion for classifying public versus private networks, whereas authorisation is the criterion for classifying permissionless versus permissioned. Detailed classification enables more appropriate usage of blockchain technology for each business requirement.

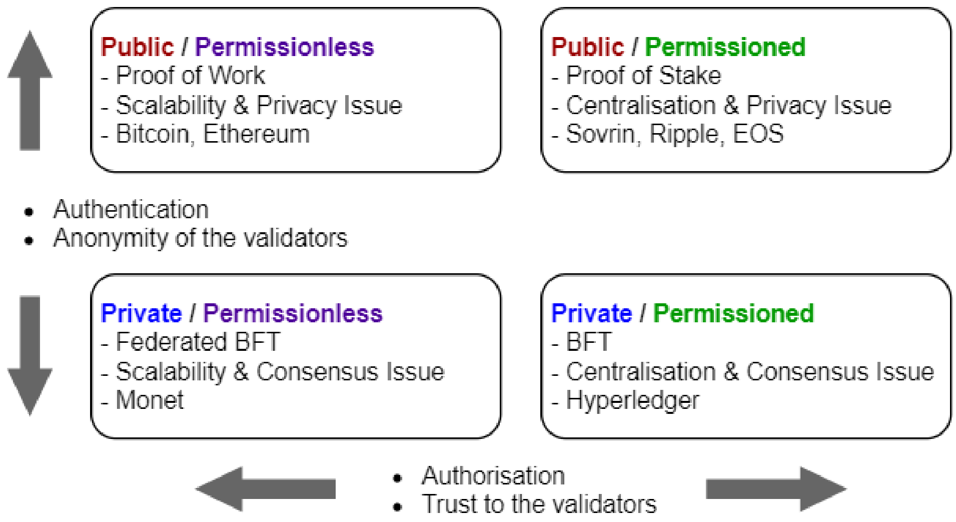

Figure 2.2: Blockchain taxonomy towards proper usages based on authentication and authorisation (reproduced from [4])

Based on the above conditions, existing popular blockchains can be generally classified into four categories, as shown in Figure 2.2:

1. public permissionless,
2. public permissioned,
3. private permissionless, and
4. private permissioned.

The well-known blockchains Bitcoin and Ethereum fall into category 1, and the representative private blockchain in the recent spotlight, Hyperledger [12], belongs to category 4, to name a few examples.

2.4 Blockchain Adoption Decision Tree

Blockchain is gaining popularity and adoption in many enterprises, as the state-of-the-art technology is often regarded as a method of strengthening competitiveness, but there is no one-size-fits-all solution to meet every requirement. In addition, in order to adopt a new technology, specific environments and business purposes should be taken into consideration first.

The most common business requirements that can be met in IoT environments are the following:

- The data generated by IoT devices should be transferred both to internal systems of individual departments in an organisation and to external business partners.
- While data are transferred to each system, data integrity should be guaranteed.
- For compliance and audit purposes, the data or transaction history should be tracked and should not be manipulated.
- Every organisation should agree on how to do business beforehand, so that a workflow should be defined and assured.
- The distributed location of each business partner or system should be taken into consideration.
- Human labour should be minimised in terms of costs and errors, although administrative work is still needed for hardware deployment.

Given the above business requirements, each case can be assessed by the corresponding decision criteria, as shown in Figure 2.3, based on the blockchain taxonomy described in Figure 2.2. By following the decision tree, the necessity of blockchain adoption in IoT environments can be evaluated.

Examining whether blockchain is necessary and which type of blockchain is appropriate can be done in the following way.

In the decision tree, an entity refers to a system node that has computational capability in an IoT environment, such as IoT devices, systems for each department, and external systems for business partners, which interface with each other.

1. As data gathered from IoT devices are processed and interfaced with multiple stakeholder entities, the answer to questions 1) and 2) is YES.

2. IoT entities include not only internal systems in the same organisation and amicable business partners, but also competitive stakeholders. In addition, they are independent from others. For this reason, the answer to question 3) is YES.

3. For operations, the predefined rules control the whole process. Once the rules have been set up, they remain in effect for quite a long time. Accordingly, the answer to questions 4) and 5) is YES.

4. Every transaction should be traceable to find the causes of problems, so transaction logging is essential, and the answer to question 6) is YES.

5. IoT devices are usually used for certain purposes in an organisation, so that they are deployed on the private network and system administrators control and manage the devices. Consequently, the final solution is a permissioned private blockchain.

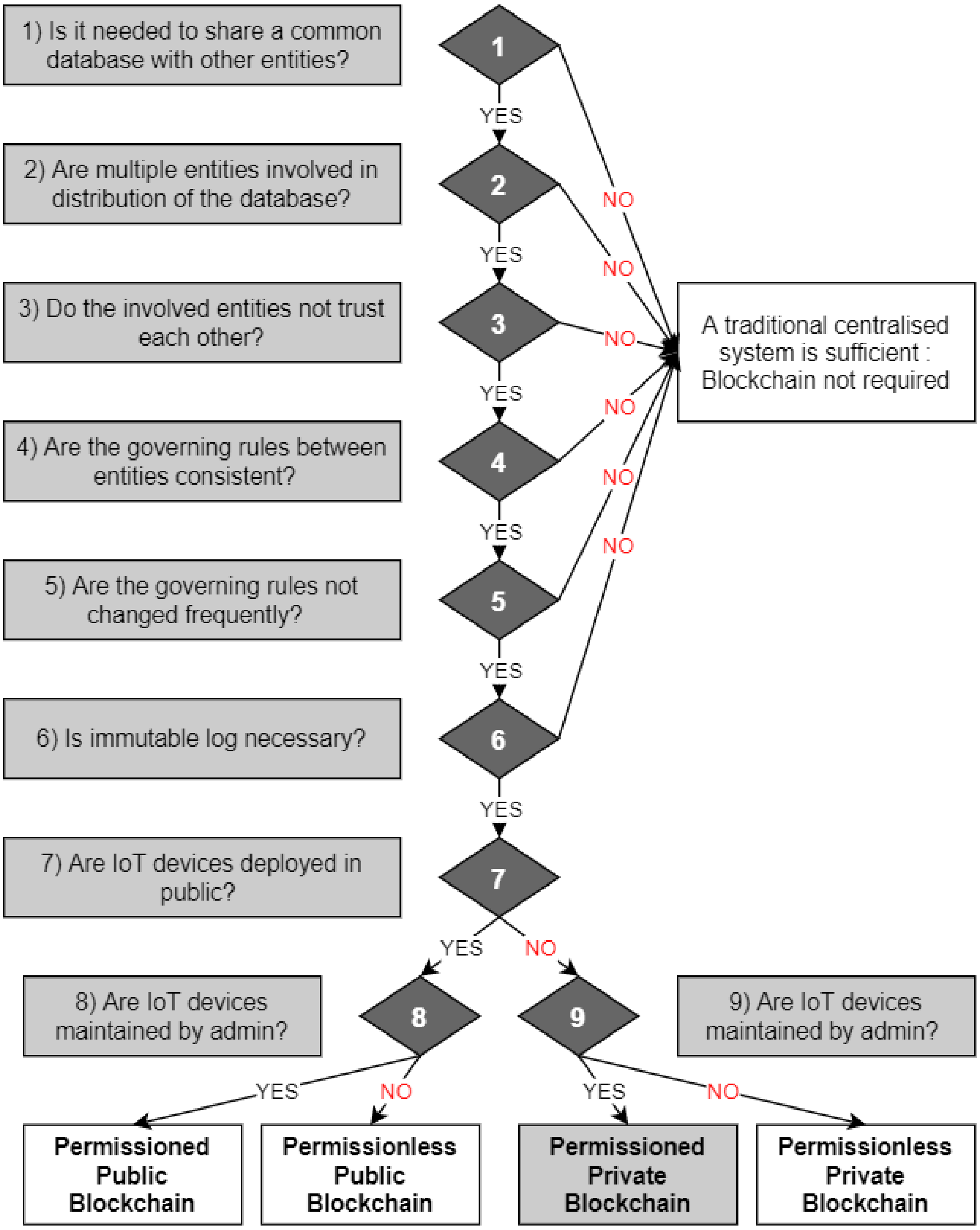

Figure 2.3: Decision tree for blockchain adoption in IoT environments (reproduced from [4])

2.5 Conventional Blockchain Limitation

This section addresses the limitations of conventional blockchains: public permissionless networks such as Bitcoin and Ethereum, which have longer histories than other blockchain types.

Although various efforts to overcome drawbacks of conventional blockchains have been made over the past several years, it is still complicated to apply new functions or upgrade the core network.

First, the performance problem is introduced in terms of transaction processing.

Second, the main cause of latency, the consensus algorithm, is addressed. In addition, hurdles in application of the conventional consensus algorithm to other types of blockchain and to IoT environments are described.

Third, data privacy issues that are inevitable due to blockchain's transparency are discussed.

Last, hard fork is described. The hard fork is rare in usage, but it eventually happens when the blockchain needs to be upgraded or modified for any particular reason.

2.5.1 Transactions Per Second (TPS)

To measure the performance of a system, the number of transactions per second (TPS) is mostly used. Although it is considerably dependent upon test scenarios and customised applications, it can be a basic scaling method to check the performance to some extent.

Generally, distributed computing accelerates processing, as it can be facilitated by parallel processing. Blockchain is based on decentralised architecture involving distributed technology. Nevertheless, blockchain has lower performance than centralised database systems [17].

It is slightly different for each measurement report, but the benchmark TPS results for different networks are as follows [18, 19, 20]:

- Bitcoin: 7 TPS
- Ethereum: 15 TPS
- VISA Card: 2,000 TPS (up to 24,000 TPS claimed)

The current TPS of the conventional blockchain is not suitable for real-time interactive transactions. In the case of IoT data processing, it becomes unacceptable.

Although various factors can affect performance, the low performance of the conventional blockchain is mainly caused by the consensus algorithm.

2.5.2 Consensus Algorithm

Proof-of-Work (PoW)

Consensus is a mechanism to achieve agreement among participants in a blockchain network in order to add or modify data. With the popularity of blockchain, it is now referred to as a consensus algorithm. It secures the integrity of data throughout the distributed ledger for all participants and protects against manipulation by malicious attackers.

As a consensus algorithm for Bitcoin and Ethereum, Proof-of-Work (PoW) is used [2][9]. PoW is based on a mechanism called 'mining'. The nodes engaged in the mining are called 'miners'. Mining is a challenging process to find a nonce value with relation to transaction validation. Successful mining proves that the miner is authenticated, and the new block created by the miner is allowed to connect to the previous block.

Therefore, the PoW consumes a huge amount of computational power and time, which is inefficient and adds latency in transaction processing.

Since blockchain was introduced, a number of different consensus algorithms have been developed. However, each algorithm has its own inherent defects [21][22]. This thesis deals with the generic concepts underlying most commonly used consensus algorithms rather than the detailed architecture of each algorithm.

Application of the Consensus Algorithm to Private Blockchain

Bitcoin and Ethereum are permissionless blockchains. Any entity can participate in the network without permission of any authority, and the entity is anonymous in the network. Each entity does not trust any other entity but regards them as adversaries.

On the contrary, in a private blockchain, participation of a new entity is selective, and the entity is mostly known to other participants. The system is operated under a shared governance. Consequently, trust can be easily attained in the private blockchain.

In the permissionless public blockchain, malicious entities can join and act in the network at any time, so there are risks, which are reduced in the permissioned private blockchain.

However, since permission and access control are needed in the permissioned private blockchain, the conventional consensus algorithm cannot be applied in the same way. Functionalities such as identity and authority management should compensate for the algorithm.

Application of the Algorithm to Enterprise IoT

In enterprise, privacy and compliance, which relate to identity and authority management, are so critical that many businesses turn to the permissioned private blockchain, while generic applications of blockchain in public are mostly permissionless. Industries operating in IoT environments demand the permissioned private blockchain.

The permissioned private blockchain does not need to rely on the mining process. The decision of using or not using the mining process is important in blockchain network development, because it affects almost every mechanism, including adding a new block, authentication, and the consensus algorithm.

Due to the sequential process of mining, the performance of the whole blockchain system is inevitably very low. Consequently, if the mining process is eliminated, although other similar mechanisms might be needed, performance is improved considerably. In addition, '51% attack' [9] or 'double spending problems' caused by malicious miners in a public blockchain no longer occur in the absence of the mining process. Another advantage in eliminating the mining process is that it reduces computation intensiveness, which reduces workload on each resource-constrained IoT device.

2.5.3 Data Privacy

Blockchain is transparent and shares all data with all participants, so that data confidentiality cannot be guaranteed. Although participants are anonymous, every participant can view all the data and transactions, which might cause serious problems.

For example, especially in enterprise, data including sensitive information, such as bidding prices, often need to be hidden against other business competitors while they are all participating in the same blockchain-based network. When organisations form consortia to serve common purposes and the total amount of the project is very large, data privacy becomes even more significant.

Since the conventional blockchain does not support data privacy for this purpose, it is not appropriate for enterprise use cases, particularly those that entail the needs of a consortium.

2.5.4 Hard Fork

Integrity and security are basically guaranteed by the consensus algorithm of all nodes joining the network, so that once the blockchain-based network is configured in a system, an upgrade or modification is nearly impossible as it can only be facilitated by the consensus. So-called hard fork is needed in this case, eventually resulting in a split from the legacy network and creation of a new one. Furthermore, the conventional blockchain architecture does not fit in IoT environments in terms of scalability and performance.

In enterprise applications, each company has its own platform and applications, which may be different from each other; even if the platform is the same, the versions of the binaries in the software library can be diverse. The bigger the business area is, the more gaps in computing environments exist. Under these distributed circumstances with great heterogeneity, software compatibility is a significant challenge in performing transactions among different organisations.

For the integrity of application services, manual labour cannot coordinate all the requests. Instead, compatibility management functions should be included in the integral parts of interfaces to support the use of different platforms and various versions of binaries. Compatibility management will eventually mitigate the need for the hard fork, but the conventional blockchain does not have such functionalities yet.

2.6 Related Work

In relation to blockchain, there are a number of studies, but very few of them address blockchain with IoT, owing to the limitations described in the previous section. For IoT data processing, cloud computing is often applied due to the large amount of data. However, in cloud computing, data ownership eventually relies on the cloud service providers. In addition, it still has defects caused by the centralised architecture, and the cost of maintenance is very high.

Research on blockchain is also focused more on the public blockchain, but the private blockchain is preferred in IoT environments as reviewed in the previous section. As a result, not all novel ideas proposed for the public blockchain can be applied to IoT environments.

In addition, numerous efforts have applied blockchain as a part of the whole system, which eventually introduces centralisation and increases susceptibility to a SPOF.

This section summarises the current studies focused on the above issues and analyses their strengths and weaknesses.

Public Blockchain Adoption and Enhancement

Since the public blockchain has a longer history than the private blockchain, most of the existing studies adopted the public blockchain to process IoT data and then sought to enhance its architecture.

Dorri et al. [23] proposed a layered network platform with blockchain. They introduced a new type of blockchain architecture for IoT that reduces both traffic and processing overheads, especially in validation of a new block, while guaranteeing security and privacy. They suggested a three-tier framework: smart home with a private immutable ledger, overlay network for higher resource devices, and cloud storage for groups of users' data. By eliminating the process of consensus and validation, PoW in conventional blockchain technology, they accelerated processing speed. Instead, they used a so-called distributed trust method, which is not described in detail.

The concept of separate processing divided into three tiers is novel but somewhat theoretical, and the model needs to be validated in more diverse IoT environments. The simulation in the paper was performed in a limited range of smart home environments. In many cases of smart home services, a service provider controls the whole system and stores the data in a remote place, which means it includes a centralised component.

Huh et al. [24] showed how the blockchain can be implemented into IoT environments. They instantiated controlling IoT devices by Ethereum smart contracts. Ethereum smart contracts were developed for a smart controller of IoT devices on Raspberry PI platform. They simulated automatic electricity supply and relevant control in an air conditioner and a light bulb model device according to a certain threshold set by a smart phone beforehand. Ethereum was used because it is turing-complete, such that it contributed to regulatory processing in the simulation as well as device authentication. Their simulation performed well enough to show the deployment of blockchain-combined IoT process. However, they paid very little attention to blockchain's defects, such as poor performance, so that Proof-of-Concept (PoC) may not be directly applicable to real enterprise environments. Moreover, the data synchronisation of IoT devices was not dealt with, even though they pointed out its importance.

Chakraborty et al. [25] proposed a novel concept of allocating computational tasks to a secondary node and configuring a multi-layered IoT network. However, they did not propose the conditions for segmentation of each layer or describe which node is located in which layer and why. In addition, the correlation between a layer and the immediate higher layer is not certain. Basically, a higher layer is derived from an existing one, and in IoT environments, gateways or routers mostly contribute to grouping of a set of nodes. Consequently, if segmentation is merely a group of nodes, the layering has no special effects. Otherwise, the purpose of the suggested one-to-many relationship is still obscure. Moreover, the limited computational power of each existing node was assumed while the validation of a new node connection was based on the blockchain consensus. Concerning high power consumption in the consensus mechanism, this validation process is inappropriate.

Cho et al. [26] described obstacles to blockchain adoption in caused by limited hardware resources while they tried to solve IoT security issues. In order to reduce overhead on IoT devices, they suggested combining cloud computing and fog computing, but without any further details. They emphasised the importance of lightweight IoT networks using the blockchain, but their analysis and proposition are vague, because a feasible blueprint is not presented. However, their proposals about fog computing and a lightweight network system for IoT with the blockchain warrant further research.

Different IoT environments, especially with the current 'smart things' technology, are well categorised and overall features of the blockchain are brief but precise in [27]. However, the proposed hypotheses of security threats are not suitable or practicable for the blockchain. For example, Denial-of-Service (DoS) attack and flooding blockchain with invalid objects are not feasible, as the blockchain is basically safe from those attacks. In addition, their architecture includes national cybersecurity agencies and companies, but these are third parties, which contradicts the decentralised concept of blockchain. In general, the description for the proposed solution falls short of explaining the procedures and the concept of the update process. The prototyping seems far from reality, since blockchain nodes are on the virtual machine and there is no differentiation in IoT devices.

Non-blockchain IoT Computing with Alternative System Components

For more efficient IoT computing environments, many studies applied additional system components apart from the blockchain platform. These works provide other perspectives on enhancing IoT data processing. However, compared with blockchain-based architecture, the alternatives are redundant and have inherent defects of centralisation.

Tayeb et al. [28] proposed a fog computing layer in an IoT computing environment. They introduced that the proposed fog computing layer has various tasks, such as aggregation of different protocols and of packet types from heterogeneous IoT devices, and security services, rather than just running as edge computing components aloof from the cloud. While these tasks are performed on an edge layer as proposed in the paper, many of them can also be implemented on a blockchain network with smart contracts, so that workloads both on edge layer and on the cloud may be reduced.

Ko et al. [29] proposed a middleware architecture for smart objects in ubiquitous computing environments. For the architecture, they used metadata of sensor devices for communication between sensors and middleware. Based on using the metadata, they suggested interoperability with sensor providers, sensor management, service management, and knowledge repository, all of which can be implemented on a blockchain network.

While the versatility of fog computing and middleware systems is taken into consideration in IoT environments, the simplicity and deployment readiness of IoT gateways with similar functionalities can be prioritised to streamline the whole system configuration when it comes to efficiency and cost. Ryu [6] presented an open-source hardware-based IoT gateway using Raspberry. The author intended to develop the architecture for an IoT open service platform, but the IoT gateway retains much of the workload, such as a local database aside from the database in the server, and the IoT data broker is segregated, which needs to be enhanced in terms of redundancy.

As for distributed computing, which is a superset concept of decentralisation, Yannuzzi et al. [30] assert that fog computing is an essential component in IoT environments, especially with cloud computing and long-distance localisation. They suggest that fog computing will enhance the adaptable and scalable platform for IoT in terms of four factors: large geographical footprint, large scale, large amount of data, and real-time analytics. With regard to this idea, blockchain and smart contracts can take many of the roles of fog computing, because a blockchain network can be located between edges and other analytic systems, and smart contracts can be utilised for localised decision-making and other operations for automation.

They also presented resource virtualisation techniques for IoT service migration. While an IoT platform shares hardware resources, it needs to maintain quality of service (QoS). For this QoS, Dusia et al. [31] proposed a job priority scheduler. Batch processing and job scheduling are ways to improve performance and can be considered in blockchain transactions as well.

As the usage of IoT devices increases, more studies have been dealing with big IoT data processing methods on cloud computing. Wang and Ranjan [32] mainly focused on IoT data-intensive workflow using big data solutions in distributed data centres. However, when data centre coverage in a wide area is concerned, one is likely to question why IoT data should be processed over the distributed data centre. In other words, in such cases, system designers should look into the business requirements first and adopt local processing in immediate data centres to reduce latency in data integration. In addition, when big data solutions are adopted, it would be closely related to data analytics, but data from IoT devices needs to be filtered out rather than aggregated for data warehousing. Whereas the approach for big IoT data processing is essential when huge amounts of IoT data are concerned, data integration in between data centres goes beyond the points. For distributed computing, blockchain based on P2P decentralisation is more appropriate.

IoT Data Integration with Blockchain

In terms of IoT data integrity on blockchain, existing studies mostly address the practical integration methods rather than focus on how to guarantee integrity.

Liu et al. [33] addressed IoT data integrity verification in cloud environments. They proposed decentralised data integrity, emphasising trust availability and auditability by using a public blockchain, Ethereum. Accordingly, the transaction fee, gas, paid in Ethereum's cryptocurrency, ether, was used to update the ledger by data owners, and the cost would deter data owners from making updates. Furthermore, processing time would be increased in IoT environments because of the large volume of data to be processed, so that data consistency in real time might not be guaranteed. It was assumed that blockchain would reach a consensus within a short time, but this would not be easy in a public blockchain.

Liang et al. [34] proposed an IoT data integration method using blockchain for communication with drones. Their experiment is a good example of data assurance and resilience in wireless IoT devices with a control system. They used cloud computing for storing and auditing data interactions between drones and the control system and a blockchain network for addressing data integrity. The data are transmitted to blockchain using a hashing algorithm, and the receipt is sent back as a response to the cloud. This ensures data integrity, but the purpose of deploying both blockchain and the cloud is ambiguous, because it may bring unnecessary replication of data, which will also cause latency in data processing. In addition, since they use a public blockchain involving a consensus algorithm for drone control, the business applications are obscure. Moreover, despite using blockchain, the proposed architecture is not able to fully facilitate decentralisation as the cloud system and the control system are vulnerable to a SPOF.

Hang et al. [35] designed an IoT blockchain platform to preserve integrity of IoT sensing data. They used the permissioned private blockchain to prove their concept, which is effective and resilient in IoT environments. Nevertheless, they deployed an additional server to control IoT devices and to send transactions, which compromised the decentralised architecture. While they applied all the functions of the private blockchain, hierarchical layer architecture and analytics module were also suggested in their architecture. However, blockchain adopts DLT, so that it is not supposed to provide big data services as it only stores the current status. Moreover, the blockchain service layer was described as though it was on the IoT physical layer. Above all, they did not consider any IoT constraints or discuss detailed methods to ensure data integrity.

Summary of the Literature Review

Table 2.1 summarises the literature review. The summary focuses on how to address IoT data integration and which technologies to use.

As can be seen in the Table, many works are dedicated to facilitate public blockchain regardless of the target application services. They concentrate on the utilisation of blockchain itself along with the paradigm of decentralisation. In order to solve insufficiency of their architecture, they use supplementary components, such as layered network, cloud computing and middleware. Some of the works are even implemented with third-party authority. These usages violate the concept of decentralisation, so that the benefits of blockchain adoption are diminished significantly.

A few works deal with IoT data processing with additional system components. When it comes to the volume of IoT data, each idea is fairly useful. Nonetheless, they do not suggest ways to maintain data integrity. Moreover, defects of centralised architecture have not been addressed.

Private blockchain has been researched actively in recent years, especially with the emergence of Hyperledger [12]. However, those works mostly deal with the core architecture of private blockchain. The cases of private blockchain application to IoT environments are still very rare.

2.7 Summary of the Chapter

This chapter presented the technological background for the proposed private blockchain-based IoT data integration platform and reviews of related work.

Relevant characteristics both of IoT and blockchain were summarised as the basis for approaching the proposed design.

Table 2.1: Summary of related works : primary research area and applied technologies

Reference	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)
Dorri et el. [23]	V			V			V	V	
Huh et el. [24]	V		V						
Chakraborty et el. [25]	V			V			V		
Cho et el. [26]	V			V				V	
Boudguiga et el. [27]	V								V
Tayeb et el. [28]				V			V	V	
Ko et el. [29]				V	V				
Ryu [6]				V		V			
Yannuzzi et el. [30]				V				V	
Dusia et el. [31]				V			V		
Ranjan [32]								V	V
Liu et. el. [33]	V		V					V	
Liang et. el. [34]	V		V		V		V	V	
Hang et. el. [35]		V	V		V	V			

(a) Public Blockchain
(b) Private Blockchain
(c) Focus on IoT Data Integrity
(d) Focus on Scalability
(e) Use of Middleware
(f) Use of IoT Gateway
(g) Use of Additional Layer
(h) Use of Cloud Computing
(i) Use of 3rd Party

The types of blockchain that exist and their application in IoT data processing were covered.

Then, the limitations of the conventional blockchain were described. This investigation was performed from the point of view of challenges facing blockchain adoption in IoT environments.

Finally, other studies of IoT data integration were reviewed with a practical lens.

The design of the proposed architecture (presented in the next chapter) is based on the knowledge summarised in this chapter.

Chapter 3

Blockchain-based Network Design

This chapter discusses the architectural design for blockchain-based IoT data integration.

First, problems and promising solutions are defined for the architecture design. The design is focused on implementation of these solutions. Then, several kinds of categorised factors are taken into consideration.

Next, the conceptual design is presented, which includes descriptions of the workflow and necessary system components.

Finally, the design specifications are proposed with a model use case.

The model system is designed to validate the proposed architecture and is used for the experiments in the following chapter.

3.1 Design Approach

This section describes issues and considerations that were taken into account before beginning to design the architecture.

Based on the problems presented in Chapter 1, issues are listed, and solutions for each issue are proposed with relevant attributes. These solutions are the key points informing design of the architecture.

Then, the design considerations are summarised into the following categories:

- blockchain and decentralised technologies,
- data processing issues, and
- IoT-specific constraints.

The proposed architecture is customised and enhanced for these aspects.

3.1.1 Problems and Solutions Declaration

Problems that should be overcome to achieve autonomous machine control in the manufacturing industry using IoT devices can be grouped into three categories, as shown in Table 3.1.

1. Autonomy is the foremost necessity in the manufacturing industry. All the machinery, facility management systems, and IoT devices communicate with each other using the data that each component produces. The complexity in configuration, the cost-ineffective labour hours, and the possibility of human errors make automatic control essential in the production line. Amongst other things, data integrity should be secured first in order to achieve autonomy.

   Blockchain, featuring data immutability, transparency, and traceability, ensures data integrity. Smart contracts can adapt the business requirements into blockchain and handle data processing autonomously under predefined conditions.

2. Despite the secure data integration on blockchain, blockchain itself has inherent drawbacks mostly related to system performance. In IoT environments, performance problems are exacerbated as huge

Table 3.1: Problem and solution elements to be reflected in design

No.	Issue	Solution
	Attributes	
1	Autonomous control for machine and equipment in manufacturing based on IoT sensor data while data integrity is guaranteed	Configure blockchain network and leverage the smart contract
	Data integrity / Immutability / Transparency / Traceability	
2	Hindrance to the blockchain adoption in IoT environments	Apply permissioned private modular blockchain model
	Transaction performance / Consensus algorithm / IoT constraints (Delay, Reliability, Ordering)	
3	Drawbacks of the permissioned private blockchain and hurdles of client integration	Streamline the configuration and develop API
	Transaction verification policy / State Database update / Concurrency control / Gateway Connection	

volumes of data are continuously generated. The performance problem is typically caused by the consensus algorithm.

Blockchain is a decentralised technology, and the consensus process by all the participants is the most significant task. In contrast to public blockchains, private blockchains can be based on a transaction verification consensus algorithm, which can alleviate the performance problem. Furthermore, the modular architecture of the private blockchain enables the system configuration to be more flexible and scalable.

3. In spite of many advantages, private blockchain has some draw-

backs. It is more difficult to implement than public blockchain. Two main reasons can explain the difficulties.

One reason is that since private blockchain is a 'private' network, membership management is needed. Subsequently, cryptographic credentials to access blockchain should be managed.

The other reason is that the consensus algorithm itself used in private blockchain involves much more interdependent configurations.

Moreover, there are discriminations between peer nodes joined in the blockchain and external client nodes. In principle, blockchain does not allow external nodes. Every node that shares data and transactions should participate in the network. However, many kinds of blockchain provide various interfacing methods to communicate with external nodes that do not join the network.

In IoT environments, IoT devices become the external nodes. Because so many devices are deployed, they cannot all join the network. Given their roles and computing resources, they are supposed to remain as external clients while they interface with blockchain.

This usage of IoT devices, instead, makes the whole system configuration complex. Additional system components are often deployed to complement the interfaces and relevant controls. Therefore, it is important to streamline the process and the configuration to maximize the effects of blockchain adoption.

3.1.2 Blockchain-specific Considerations

Enterprise-grade Private Blockchain Type

Amongst well-known permissioned private blockchains, Hyperledger Fabric [12][36] is the most commonly used, especially in enterprise.

Hyperledger Fabric is one of the open source-based collaborative Hyperledger projects hosted by Linux Foundation. Hyperledger Fabric boasts modularity and scalability and is especially suitable for IoT environments. Since its first version 1.0 was released in 2017, several hundred developers have been working on Fabric blockchain core; versions 1.4 LTS and 2.0 are currently supported. It features the following main services:

- membership service provider for privacy and permission,
- channel service for private data process,
- ordering service for block aggregation,
- endorsement for transaction verification, and
- pluggable consensus algorithm.

Chaincode Usage

There are two types of chaincodes in Hyperledger: One is a system chaincode for the interaction with the given blockchain, and the other is a user chaincode that users or developers create for the business logic. By developing this user chaincode, which is referred to as a smart contract in other blockchain types, the data processing algorithm can be implemented on blockchain.

Decentralised Configuration

To overcome the challenges facing blockchain adoption, a variety of methods have been used. However, most of them involved additional computing resources, such as middleware, controllers, brokers or cloud computing [33, 34, 35, 37]. In such cases, although advantages of blockchain have been exploited, the architecture introduces disadvantages of the centralised system, such as a SPOF.

In addition, while additional resources are deployed to improve effectiveness, they increase complexity of system configuration, requiring more administrative costs. In terms of flexibility and maintenance, such architecture requires much operational labour and even service downtime when reconfiguration is needed. Therefore, streamlined system configuration is required, while additional resource deployment is minimised and decentralisation is reinforced.

3.1.3 Data-centric Considerations

Transaction Data Processing on Blockchain

Blockchain is essentially aimed at both transaction validation and logging by distributed participants, so that all the transaction data might not be stored in blockchain even though it can be. However, when it comes to the huge amounts of IoT data that are generated continuously, the necessity of an effective way of processing the data arises. Hence, the following aspect should be considered.

All the data gathered by IoT sensors should not go through the verification process conducted by participating nodes on blockchain, which would cause significant performance degradation. To reduce the amount of processed data, only selective data filtered by predefined threshold or criteria should be transmitted into the blockchain via the IoT gateway. Among the huge volume of sensor data, very little has special meaning to operations.

Data Preservation for Compliance and Audit

While efficient IoT data integration into blockchain is an area of focus in enterprise applications, some businesses are under strict regulations and require compliance in terms of data preservation. In those cases, IoT sensor data should be not discarded, but should be stored somewhere, either in the IoT gateway or the blockchain. However, both are not appropriate as far as storage capability and performance are concerned. In most cases, cloud storage is used, which costs extremely high only for that purpose. In addition, cloud computing might create a bottleneck or a SPOF, which goes against the concept of decentralisation. To solve these problems, InterPlanetary File System (IPFS) [38] is a promising alternative with lower cost and decentralised features.

Distributed and Clustered File Storage

IPFS is a P2P distributed file system in the form of a single BitTorrent swarm using content-address and Merkle tree-based directed acyclic graph (DAG) [39] access method; it is fast (due to proximity in file storage) and resilient. It is a versioned file system similar to Git [40]. In terms of file distribution and user experience, it is better than the current HTTP, which relies largely on web browsers [41] and location-based file search. In this regard, IPFS is a so-called 'permanent' web based on decentralisation.

Sensor data are transmitted continuously and periodically from IoT sensors to IoT gateways, and according to predefined criteria, can be processed further or discarded. However, when there are compliance requirements, all data should be forwarded to IPFS.

3.1.4 IoT Constraints

Latency in Data Transmission

IoT data are generated and sent in real time, but a variety of factors can delay data transmission. If the latency is caused by an IoT device, most of the other devices are alive to send the data on time, and the whole process is unaffected. Otherwise, in case of network problems, all the sensed data are likely to be delayed. To compensate for this, network routing can be separated. In blockchain, this separation can be facilitated by P2P connection.

The most important thing in this regard is that indispensable information must be processed. How to deal with the delayed data is the question. The network path from IoT sensors to the main systems is also crucial.

Sensed Data Congestion

IoT sensors far outnumber device management or control systems for a number of reasons. Above all, IoT sensors are cheap and lightweight so they are deployed as needed to gather sufficient environmental information accurately. When sensed data generated by each sensor are transmitted simultaneously, the massive volume of data causes congestion that becomes a significant overhead to the system.

However, thorough congestion control may cause data overload to the network, which also decreases performance [42]. Therefore, more efficient and streamlined data processing logic should be involved. The system should be able to guarantee high performance to a certain degree depending on the business requirements, and concurrency control is essential.

Vulnerability to Attack

Due to their hardware resource constraints, most IoT devices are not equipped with security mechanisms and are therefore vulnerable to attack. Trustworthiness and authentication of IoT devices should be considered to allow the devices to access blockchain. For security and efficiency in the whole system configuration, IoT devices should remain as external clients interfacing with the blockchain. In addition, data integrity checks should be required.

IoT sensors, in particular, send data via a certain medium. Because they do not receive the transaction result, but only send data one way, IoT sensors do not demand computational resources except for the data transmission. They have only to be coupled with the medium in a secure way.

3.2 Conceptual Design

Based on the considerations that should be taken into account before designing, this section describes the conceptual architecture.

In a bid to propose the architecture, applicable objects are defined and then the design is illustrated.

The objects are extracted from a high-level view of a system landscape in a manufacturing production line, and the whole landscape is presented first. Later in this section, the essential system components for the proposed architecture are introduced.

3.2.1 Enterprise System Landscape

To create a high-level perspective of the architecture design, the whole system landscape must be considered. Figure 3.1 shows a conceptual enterprise blockchain system landscape for an IoT-based production line in a manufacturing industry. As pointed out in Section 3.1.1, a private blockchain is selected for this use case.

It includes IoT devices to detect environmental state information and external stakeholders such as hardware parts suppliers, logistics companies, and outsourced technical support teams. All the entities interact with each other and share information on the blockchain.

- The blockchain network is initiated with several discrete channels. Each channel represents different purposes, and entities are connected with one or more channels.

- A peer is a basic entity that participates in the blockchain. Among peers, there exist special nodes: anchor peers, endorsers and orderers. Anchor peers are representatives that propagate information of each peer, such that at least one anchor peer exists in every organisation. Endorsers verify transactions, whereas orderers adjust transactions sequentially.

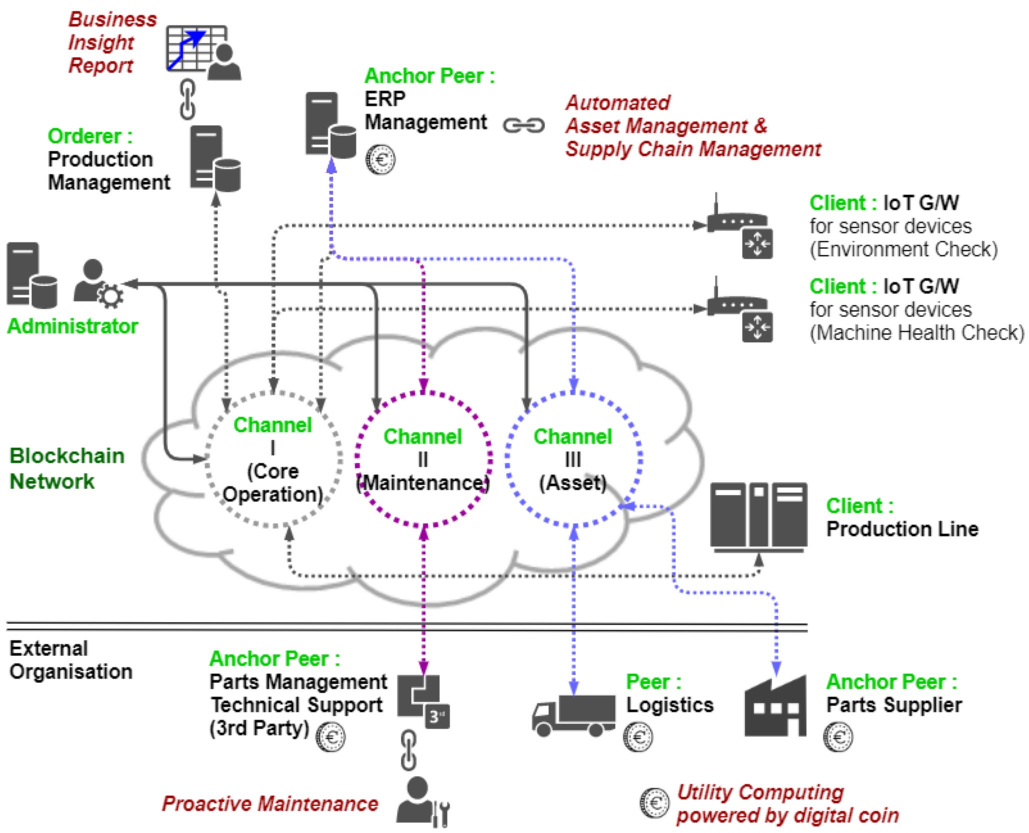

Figure 3.1: Conceptual enterprise blockchain system landscape

- Except for orderers, all the peers on the blockchain share the ledger database and the blockchain. They replicate the transaction and configuration data locally.

- The entity is identified as a client if it interfaces on transactions with the network but does not join the network. Clients are IoT devices such as IoT sensors, IoT gateways, actuators, and other devices with mobility to send data frequently. IoT sensors notice environmental states and send them to IoT gateways. Then, the IoT gateways submit transactions to the blockchain, the results of which invoke actuators' action.

- Since it is a private blockchain, administrative tasks are inevitably involved. However, those tasks are limited to such things as initial network configuration, membership management, and network channel update.
- Given data on the blockchain, hardware parts delivery and management processes are autonomously invoked. Electronic payments are realised using digital coins, which reinforces utility computing and automated settlement.
- Immutable digital records provision business insights into manufacturing operations, one of which enables proactive maintenance services.

3.2.2 Data Flow

Under the conceptual circumstances seen in Figure 3.1, the data flow in the blockchain core operational part is depicted in Figure 3.2. The proposed workflow focuses on sensor data processing to control corresponding actuators.

1. IoT sensors gather environmental data such as temperature, humidity, power supply, water supply, vibration, and dust density, which they continuously transmit to the geographically nearest IoT gateways or tightly coupled IoT gateways in signal.
2. IoT gateways collect and filter the data from sensors and forward appropriate information to control facilities for machinery. IoT gateways are authenticated by Certificate Authority (CA) using X.509 certificates, which are issued when they are enrolled to the network, so that transactions from gateways with certificates can be authorised.

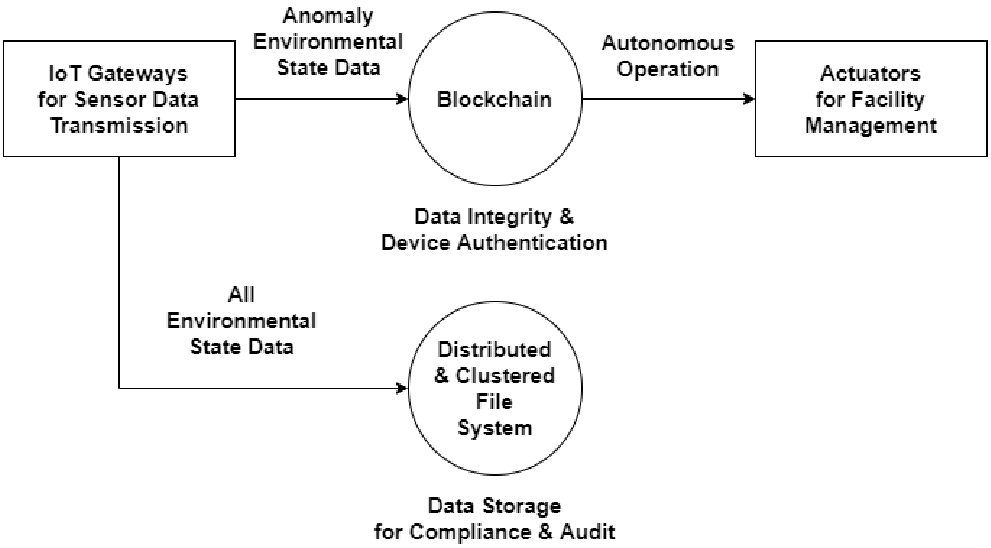

Figure 3.2: Conceptual workflow of the proposed architecture

3. IoT gateways upload all the data to distributed and clustered P2P storage systems to support compliance and the audit trail of an organisation.

4. IoT gateways also transmit sensed data to the blockchain at the same time. The gateways send the data periodically to assure seamless communication with the blockchain. The data are filtered by gateways according to the predefined thresholds to ease the workload on the blockchain. Only anomalous data are processed further. The selection criteria, however, are not strict, as more sophisticated procedures are applied on blockchain. The loose condition reduces the need to reconfigure the IoT gateways, which effectively reduces maintenance costs.

5. Once a transaction from the gateway has successfully completed, the operational status of an associated actuator is updated on the blockchain and the result is sent to the actuator application.

6. Finally, facility management systems take proper actions to cope with

environmental variation, based on results from the blockchain. The facilities are operated by actuators, which are also IoT devices. Applications on the actuators listen to the transaction results from the blockchain at all times so as to respond swiftly.

7. Data from gateways are processed and integrated on the blockchain, so that the transaction history may be tracked and immutability and transparency are ensured.

3.2.3 Major Components

Based on the workflow scenario seen in Figure 3.2, the system configuration is elaborated as shown in Figure 3.3. In this architecture, the system essentially consists of the blockchain, distributed and clustered file storage, IoT sensors, IoT gateways, CA server, peer nodes, actuators, and manufacturing machinery.

Blockchain and file storage are described in detail in Section 3.1 and Section 3.3.

IoT Sensor

IoT sensors can collect information about their surroundings, including temperature, humidity, fire conditions, and so on. They transmit sensor data through wired and wireless communication to IoT gateways. The wireless coupling of IoT sensors with IoT gateways depends upon the strength of the signal. Thus, IoT sensors and gateways are grouped geographically for better management. Coupling is established to ensure seamless availability.

To address the previously described security issues inherent to IoT sensors, interactions between IoT sensors and gateways are restricted. IoT sensors cannot access the blockchain, because they are not issued certificates, but transmit data only to IoT gateways. IoT sensors lack the authorisation to access the blockchain.

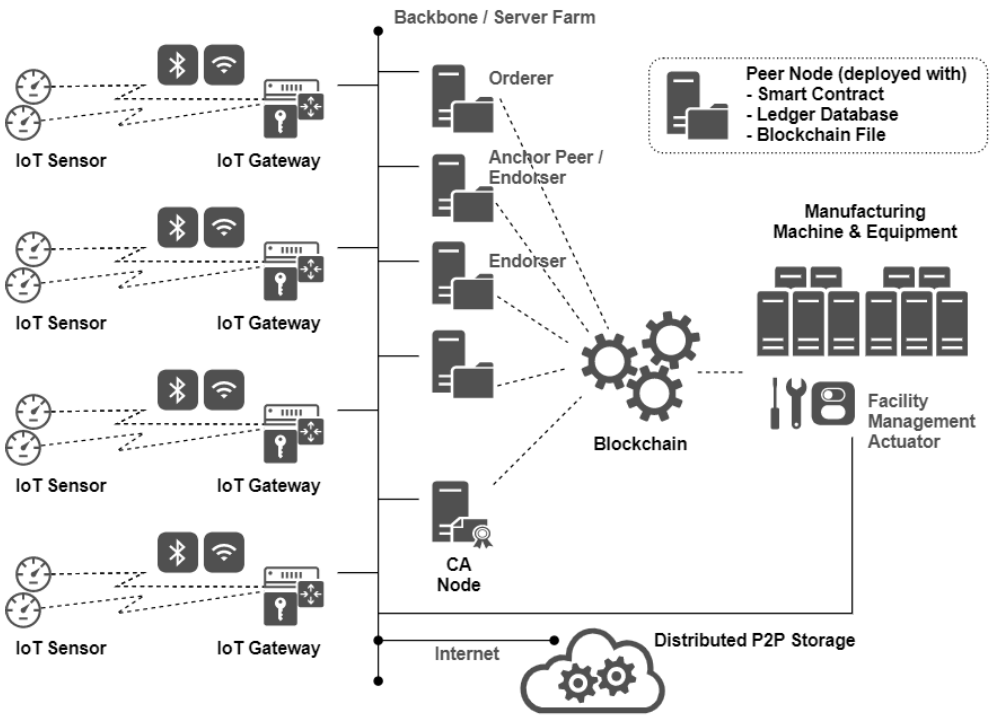

Figure 3.3: Major system components of conceptual architecture

IoT Gateway

In practice, the IoT gateway plays an important role.

First, it filters the data from IoT sensors, reducing the network overhead.

Second, it enables interoperability among diverse communication protocols from heterogeneous devices in different environments, such as 6LoWPAN, ZigBee, RFID, Bluetooth LE, NFC, SigFox, Cellular, Z-Wave [43] and Ethernet.

Third, it manages IoT sensor devices for predictive maintenance by firmware updates and health checks and to authenticate the devices.

Lastly, but most importantly in the proposed architecture, as a client, the IoT gateway processes and integrates IoT sensor data onto the blockchain using a client application Software Development Kit (SDK).

In order to promote adoption of blockchain in IoT environments, all the IoT gateway capabilities should be utilised. Then, the streamlined configuration can accelerate essential processes effectively and mitigate many known hurdles by eliminating the need for additional system components, such as middleware or other intermediary servers.

If malicious attacks or compromised data occur in the IoT sensors, only the immediate gateway connection needs to be pruned on the blockchain. Accordingly, the security of the whole system can be guaranteed.

If an IoT gateway itself is contaminated, interaction with it is denied by the blockchain because IoT gateways use encrypted authentic public and private keys to access the network.

CA Server

A CA server can be implemented by the organisation or can be used with a third-party solution. When a new IoT gateway needs to join the network, it is registered by a technical operations team on the CA server. Then, the CA server provides an X.509 certificate to the IoT gateway, which is stored on the local file system or other secured hardware medium for authentication when a transaction is invoked.

Peer

Peers are the core components of the blockchain. They host smart contracts and the blockchain network. Replicas of the current data state and transaction history are distributed to each peer in the form of state database and blockchain file, respectively. Endorsement, transaction ordering, and other interactions, such as gateway services, among each component of the whole system are run on one or more peer separately or on the same peer.

In enterprise applications, each organisation can have one or more anchor peers to gather the information of other peers by using the Gossip protocol [36], so that any peer has only to request information from anchor peers if it needs another peer's identity. Endorsers and orderers are on multiple peers in order to improve transaction processing performance by using parallel processing and to ensure fault-tolerant processing.

In practice, peers can be implemented mostly on Docker containers [44], which reduces deployment and operational costs. Furthermore, using Docker containers eliminates the need to consider the binary compatibility of heterogeneous environments.

Actuator

In the production line, there exist manufacturing machinery and control systems. Control systems involve a number of peripheral systems, such as a console, a monitoring dashboard, and a facility management system. Facility management systems are operated with actuators. The autonomous operation of these actuators based on IoT sensor data is the aim of the proposed architecture.

As in the case of IoT gateways, actuators are clients in the blockchain's view, so that they should be authenticated and authorised by CA servers.

Manufacturing Machinery

The above components enable stable production and autonomous operation in the manufacturing machinery. The machinery does not necessarily connect to the blockchain network, as sensed environmental data have no direct interfaces with the machinery.

Accordingly, the machinery can run independently of the blockchain, which lessens the complexity of system configuration in the factory.

3.3 Model System Design

This section presents a model system design as a PoC based on the concept discussed in Section 3.2. Compared to the design in the previous section, this section takes a more practical approach.

First, the transaction flow describes more specifically which component conducts which tasks in sequence. In the proposed flow, essential system components for the PoC are illustrated.

Second, a physical system architecture for the PoC is outlined, showing how to deploy each component physically, inclusive of applied technologies.

Third, processes that take place in the smart contract on blockchain for the PoC is explained. This subsection deals with the algorithm in the smart contract.

Last, the client application and its integration with blockchain are described. There exist two types of clients for the PoC: the IoT gateway and the actuator. These clients are not participants in the blockchain, and they must interface with blockchain in a certain way.

3.3.1 Transaction Flow

Figure 3.4 depicts the workflow in detail, describing each task on each component with the executing orders.

It is assumed that every IoT gateway is enrolled to CA servers (i). For this, an internal CA server is used. For the enrolment, the CA server provides an X.509 certificate (ii) that is stored in the file system on each gateway in the forms of cryptographic public and private keys (in this architecture, a digital wallet).

1. IoT sensors transmit data periodically to IoT gateways.
2. IoT gateways upload the sensed data to IPFS to preserve it for compliance and audit.

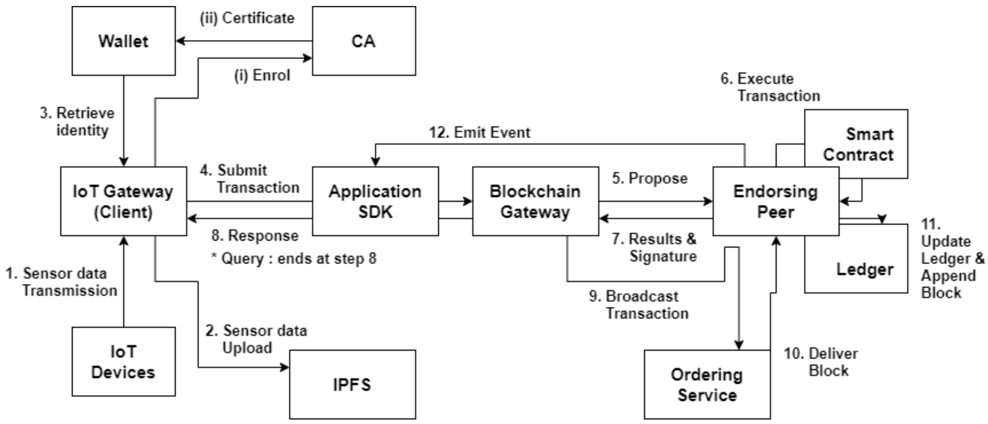

Figure 3.4: Transaction flow

3. When abnormal environmental data are detected by an IoT gateway (for instance, when the temperature goes over the predefined threshold upper limit), the IoT gateway triggers the relevant transaction, which is defined in a smart contract on the blockchain. First, to invoke the transaction, it retrieves the certificate from its digital wallet to be authorised by the blockchain.

4. The IoT gateway, as a client on the blockchain, then submits the transaction using the deployed application SDK.

5. The application SDK sends a propose message for the transaction to endorsing peers defined in the blockchain gateway API's connection profile through the gateway.

6. The endorsing peers execute the transaction using already deployed smart contracts on the blockchain. At this time, the data do not update on the state database or the blockchain.

7. The endorsing peers send back the results of the transaction, including their signature if the transaction is validated, to the application SDK.

8. The application SDK forwards the results from the endorsing peers to the IoT gateway. In case of data query rather than update in general usage of the blockchain, the process ends at this step.

9. Once the application SDK receives the results and the signature, it broadcasts the transaction to be ordered.

10. After the transaction is ordered and a block is assembled by the ordering service, the block is sent to committing peers.

11. The committing peers update the state database and append the new block to the blockchain.

12. Finally, the committing peers emit the event to the application SDK to notify that the transaction has been processed.

3.3.2 Physical System Architecture

Figure 3.5 depicts the physical architecture of the PoC.

Linux is installed on a server machine as the operating system. This Linux server represents the whole blockchain-based system of an organisation based on the proposed architecture. The whole system is streamlined with only essential components.

IoT sensor data generation is simulated, as the design focuses on data integration on the blockchain rather than on gathering sensed data from sensors. It is assumed that all the data from IoT sensors are transferred to IoT gateways.

The IoT gateway uploads all the sensed data to a public decentralised P2P file system, IPFS. With minimum criteria, the IoT gateway separates data that need to be processed further from ordinary data, which are discarded. All the sensor data are stored in IPFS, even though the normal data with the usual state are discarded in the IoT gateway.

The most popular enterprise-grade private blockchain, Hyperledger Fabric, is implemented to verify the proposed design. As Hyperledger

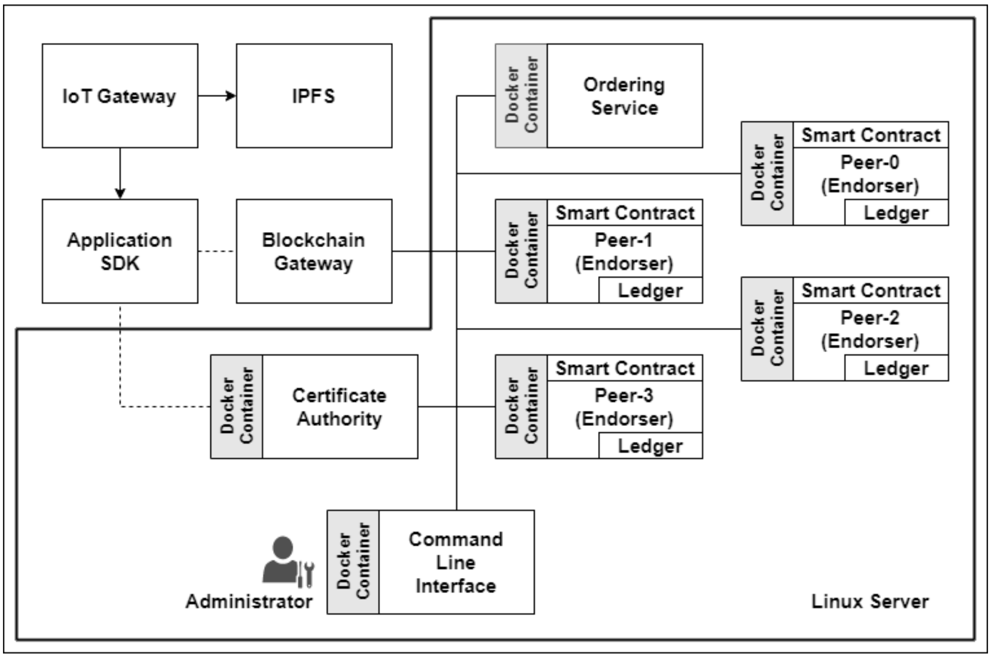

Figure 3.5: Proof-of-Concept (PoC) system configuration

Fabric is available as Docker [44] images, Docker containers host each system component with relevant software libraries. Docker is a set of running software that includes application codes and all the dependencies, so that developed applications can be deployed quickly and reliably to other platform regardless of the environments in which they run. It is regarded to be an enhanced and lightweight virtualisation concept. The set of packages is distributed in the form of Docker images.

On top of the Docker container images, platform-dependent executable binaries from a Hyperledger Git repository generate initial materials, such as channel artifacts for blockchain network credentials and the genesis block to which blocks are appended. The Hyperledger Fabric software library is available both for native code and for Docker containers. Regarding readiness of installation on diverse environments in practice, Docker container-based installation is preferred. This enables each component to be deployed faster and configured more easily.

In addition to the conveniences, each Docker container represents each physical machine virtually, such that real environments, where each node operates independently, can be simulated easily. The nodes on each container organise a Hyperledger blockchain network channel and communicate on it.

For the consensus for transactions, there must be an ordering service. When it comes to decentralisation, there should be multiple orderers. This concept is fairly intrinsic as it is based on the consensus of participants. Although there will be multiple orderers in practice, the PoC places one orderer only to confirm its ordering operation. In the case of peer nodes, however, multiple nodes run on discrete Docker containers. The peer node represents a medium on blockchain processing IoT sensor data that are transmitted from IoT gateways. In order to receive data from a considerable number of IoT gateways located in each sector, multiple peer nodes exist. These peer nodes are participants in the blockchain, while IoT devices are not. Peers elected as endorsers verify transactions and decide whether or not to acknowledge them. The multiple peers also enable load balancing and fault-tolerant processing.

Each peer hosts a ledger replica and smart contracts. As the state database, a NoSQL key-value store structure database is dedicated to each peer node. The block of the network is stored on the local file system of each peer node, which is a disk volume of each container.

A CA system provided by Hyperledger provides the certificates to authenticate each IoT gateway on the network. It runs on an independent container.

Administrative tasks on the system console are done through the command line interface container. These tasks involve network configuration, system operations such as manual update, and tracing logs. Hyperledger binaries can be run on these containers.

3.3.3 Transaction Processing on the Blockchain

Figure 3.6 shows the actuator control procedure in the smart contract for the PoC by using a state update function with IoT sensor data as an input.

1. The IoT gateway transmits sensor data to the blockchain by a client application through the blockchain gateway. Then, the data are processed by a smart contract. The smart contract basically checks input variables.

2. Input data integrity is validated by a hash function, SHA256. Input data accompany the hash value in its tail, generated by its data combination composed of IoT gateway ID, actuator ID, sensed timestamp, and sensed data signal. In the smart contract, the hash value is calculated again with only the data part. Then, the input hash value and the calculated hash value are compared, so that data integrity is guaranteed during transmission under harsh network environments. If both values are equal, the next step is executed; otherwise, the transaction is revoked, returning an error message.

3. Next, sensed time is compared with the current system time as a latency check. Data that are delayed beyond a permissible interval are discarded. This returns an error and the transaction ends. The data do not require further processing since they are out-dated and no longer valid for actuator control. The actuator should react to the current environment.

4. After data validation, the smart contract decides whether the actuator should be enabled or disabled. First, it retrieves the current actuator status. Based on the prescribed threshold minimum and maximum values, it compares the retrieved state with sensed data signal. When the actuator should be enabled but is currently disabled, the smart contract updates the actuator status to 'enabled' in the state database. Since the smart contract can only return the transaction ID

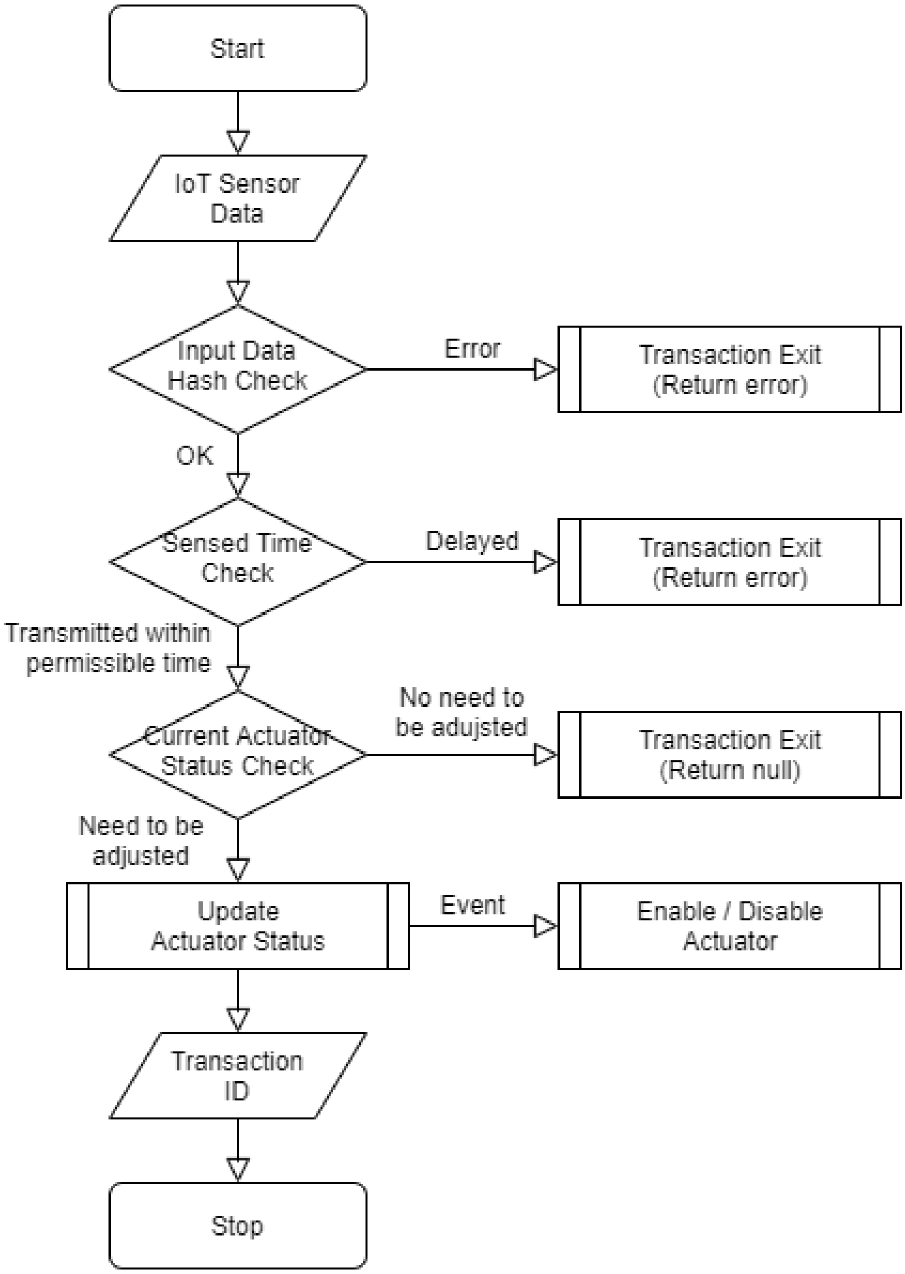

Figure 3.6: IoT sensor data processing algorithm flow chart

as a successful result of a transaction, it emits an event to associated application clients or actuators for further processing. Inversely, the actuator status can also be updated to 'disabled' based on the comparison of the retrieved state with the sensed data signal.

(a) Suppose that several IoT gateways, which handle the same actuator, send data regarding the same action of the actuator. If one single gateway sends the wrong signal and the signal is processed, it may cause a false operation. However, in this case, IoT sensors will detect the current environment state and send the correct signal immediately and operation will be corrected automatically. Since multiple IoT sensors and gateways are deployed in practice for accurate measurement and fault tolerance, and most devices are likely to send the same signal, it will be processed regardless of any faulty signal.

(b) If a signal is sent but the actuator status does not need to be changed by the condition, the smart contract only returns a success code message. This also ensures that the duplicated data signal does not affect the actuator control, so that the state database update does not happen, which reduces unnecessary updates.

(c) Regarding the update process in blockchain, every update transaction changes the state at every time, because the blockchain records all transactions. There is no duplicate update check. Consequently, if the same signals are continuously sent, the actuator status is updated repeatedly. Eventually, only the last update is the decisive one. Many conventional smart contracts follow this update method. However, such algorithms require high computing resources, since there are frequent updates. The update transaction goes through endorsement along with ordering, so the cost may be enormous. The duplicated data processing method proposed in (b) offers better performance and less possibility of transaction failure than the conventional update method.

3.3.4 Client Application Integration

Clients

Clients in the PoC are IoT gateways and actuators. IoT gateways transmit sensed environmental data to both IPFS and blockchain. Actuators are the controllers in a factory for the facilities, acting appropriately according to the result of blockchain processing.

They are not participating peer nodes on the blockchain, so there should be an interface method in order to process transactions to and from blockchain.

Digital Wallet

Once a client node is registered and enrolled in the CA system, it receives credentials to access the blockchain and to submit transactions with an authorised signature. These credentials consist of a certificate, a pair of public and private keys in a cryptographic form. They are stored in a so-called digital wallet.

The digital wallet can be implemented selectively using several methods, such as local file system, in-memory, hardware security module (HSM), and local database system. The local file system is used for the PoC as it is easy and can be mounted anywhere on the network.

Client Application SDK and Blockchain Gateway

Hyperledger Fabric provides the SDK that allows clients to interact with the blockchain [45]. The client application SDK leverages APIs to access the blockchain and to submit transactions.

The blockchain gateway provided by Hyperledger Fabric is an API module that enables client applications to interact with the blockchain. Once the client applications have been connected with the gateway, the gateway manages all the subsequent interactions on behalf of the client applications based on the pre-defined configuration.

Thus, the client application initiates transactions with the following procedure:

1. create a wallet class and retrieve credentials,
2. create a gateway class and identify an access point (peers) with credentials,
3. connect to the network,
4. access the smart contract, and
5. submit the transaction.

3.4 Summary of the Chapter

In this chapter, a number of relevant considerations for the design were first presented. Then, based on these considerations, both conceptual and physical designs were proposed.

For the proposal, IoT sensor data integration for control of an autonomous facility management system was suggested, and a brief workflow was described.

With this scenario, the specific model architecture was depicted.

The design proposed in this chapter is validated through various experiments in the following chapter.

Chapter 4

Validation

This chapter describes validation environment and evaluation methods used in the experiments. A model system for a manufacturing automation use case is presented. The model includes both blockchain and the client application in IoT gateways.

Since the most popular public blockchain platform, Ethereum has been applied in many use cases in the industry, the differentiations with it from the practical perspective are additionally addressed in the last part of this chapter.

4.1 Validation Environment

In this section, the validation environment is presented. Firstly, the development environment for blockchain and client applications is specified. Secondly, the test system configuration is presented. Lastly, evaluation methods are explained.

4.1.1 Development Environment

Blockchain Platform

Table 4.1 presents the technical specifications of the development environment for the proposed architecture. The server is comprised of Intel i7-6700 @ 3.40Ghz CPU and 8GB main memory. The operating system is Ubuntu 18.04.3 LTS. For the private blockchain package, Hyperledger Fabric 1.4.4 LTS is used. Hyperledger Fabric uses Docker technology to deploy peer nodes and to configure the network. Docker engine is based on 19.03.5 along with Docker Compose 1.25.0 and Docker Machine 0.16.0.

Table 4.1: Blockchain development environment

Component	Specification
CPU	Intel i7-6700 @3.40GHz
Main Memory	8GB
OS	Ubuntu 18.04.3 LTS
curl	7.58.0
docker engine	19.03.5
docker compose	1.25.0
docker machine	0.16.0
node	10.19.0
npm	6.13.4
Hyperledger Fabric	1.4.4
go	1.13.5
IDE	MS VS Code 1.44.2
VS Code Ext #1	MS Go 0.14.1
VS Code Ext #2	MS Remote SSH 0.51.0
VS Code Ext #3	IBM Blockchain Platform 1.0.28

The local client applications such as actuator control functions are written in Node 10.19.0. The command line tool for the data transfer, cURL 7.58.0 downloads installation binary files. The chaincode is written in Go programming language 1.13.5. Microsoft Visual Studio Code 1.44.2 is used as the integrated development environment (IDE). On the VS code, the following extensions are added : Microsoft Go 0.14.1 for the basic syntax check, Microsoft Remote SSH 0.51.0 for the remote access to blockchain system, and IBM Blockchain Platform 1.0.28 for the unit test of developed chaincode functions.

All the software and packages are the latest at the beginning of the validation. Some of them are released with higher version later, but those used in the validation were the most recent stable versions with the long-term support.

IoT Gateway Client Application

The client is an IoT gateway. For the IoT gateway, Raspberry Pi 3 is used. Raspberry Pi is an open source based small computer with a single board. It is widely used as an IoT device since it is lightweight and versatile [6]. The development and evaluation environments are as shown in Table 4.2 and 4.3 respectively.

The CPU is equipped with Broadcom BCM2837 Quad Core @ 1.2GHz and the capacity of main memory is 1GB. The OS is Raspbian GNU/Linux 10. The OS and other relevant software are installed on external SD card. In order to develop client applications, Node 10.19.0 is used along with the client software development kit (SDK) for Node.js 1.4.8 provided by Hyperledger.

To avoid unnecessary installation of an IDE on the host, which is the IoT device, ATOM 1.45.0 is used as an IDE with the FTP remote access package 0.18.0. The developed objects are transferred from the local editor to the host. In the case of MS VS code, software libraries for the IDE configuration and extensions should be installed locally on the host. In practice, it is necessary to consider the software distribution. The above development method can be applied as well. For the client application development, Node.js is used, and it does not need the software build process, so that the developed applications can be deployed by FTP. All the developed source codes are available at Appendix A.

Table 4.2: Client application development environment

Component	Specification
Model	Raspberry Pi 3 Model B
CPU	Broadcom BCM2837 @1.2GHz
Main Memory	1GB
Wireless LAN	BCM43438 2.4GHz and 5GHz
Ethernet	100 Base
OS	Raspbian GNU/Linux 10
curl	7.58.0
node	10.19.0
npm	6.13.4
Hyperledger Fabric SDK for node.js	1.4.8
IDE	ATOM 1.45.0
ATOM package	ftp-remote-edit 0.18.0

The client applications are developed on the IoT device to make it as close as possible to the real environment, but for the simulation of hundreds of devices, the experiments are performed on a different workstation as shown in Table 4.3 which is capable of parallelised load tests.

The installed software for the test is the same as in the development environment. This simulation host is connected to the blockchain network via ethernet. The simulation host and the blockchain host are located in adjacent but different buildings using the same backbone network.

Table 4.3: Client application simulation environment

Component	Specification
CPU	Intel i7-6700 @3.40GHz
Main Memory	8GB
OS	Ubuntu 18.04.3 LTS
node	10.19.0
npm	6.13.4
Hyperledger Fabric SDK for node.js	1.4.8
IDE	ATOM 1.45.0
ATOM package	ftp-remote-edit 0.18.0

4.1.2 Test System Configuration

Blockchain Configuration

The test private blockchain system has been constructed using Hyperledger Fabric on the development server as can be seen in Figure 4.1.

Hyperledger Fabric is basically targeted for enterprise applications, so that when a group of participants on the network needs data privacy against other participants, Hyperledger provides a private data channel for the group. Though multiple channels can be configured in Hyperledger for the business requirements, only one channel was configured as the scenario under consideration does not organise the consortium in the proposed architecture. Four peer nodes were deployed using Docker container with one organisation scenario.

Hyperledger Caliper Installation

Hyperledger Caliper [46] is a benchmarking test tool that measures the performance of blockchain core. It is installed and run on the blockchain development server.

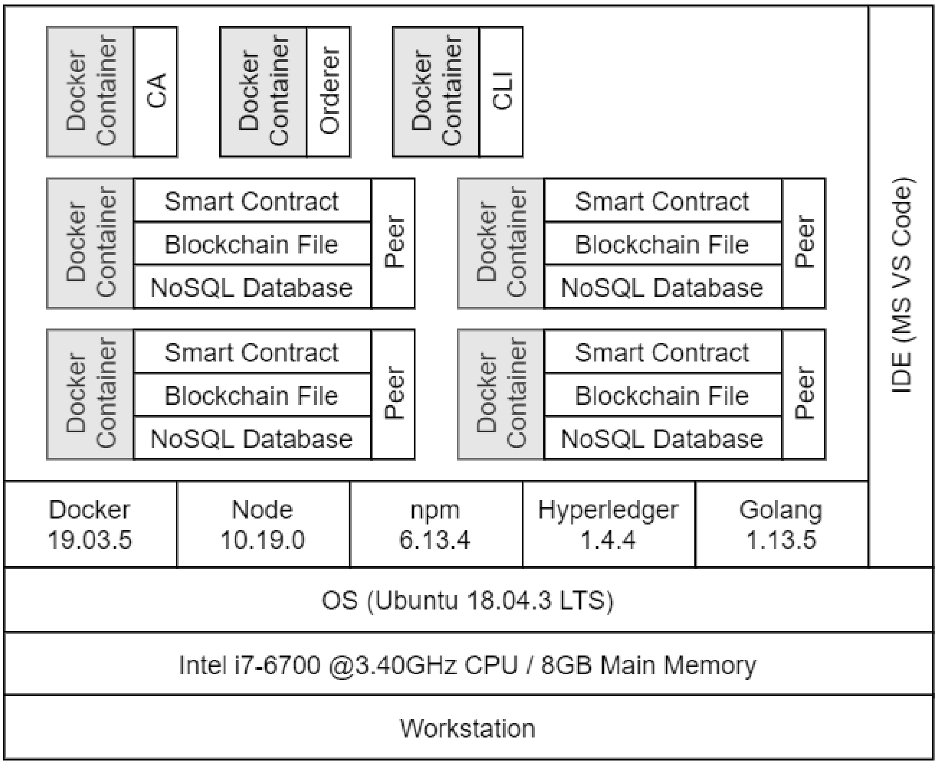

Figure 4.1: Test blockchain system configuration

By using Hyperledger Caliper, the latency of a transaction can be measured. The simulation was performed by increasing client numbers and controlling the loads at a certain time interval in the Hyperledger Caliper workload configuration. The latency and system utilisation are reported as shown in Figure B.1 in Appendix B.3.

Client Simulation Configuration

As for the client application experiments, two steps were taken. Firstly, the client applications were developed on the IoT device as shown in Figure 4.2, and tested all the functions on the device including blockchain communication. Secondly, all the developed materials were migrated to a different workstation as in Table 4.3 for the simulation.

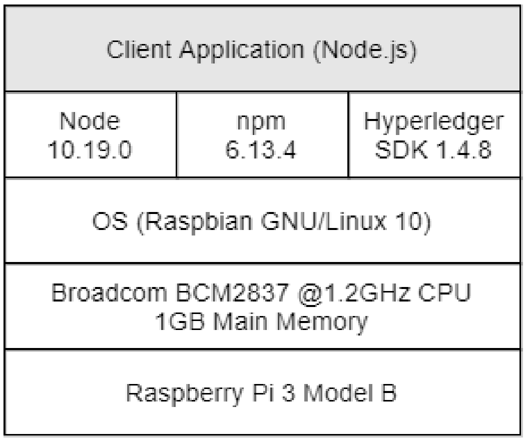

Figure 4.2: Test client system configuration

Each IoT gateway merely submits one single transaction at a time with the sensed data only from the sensors that are attached to the gateway. Accordingly, IoT gateways are unconstrained from hardware resources, so that the simulation for the multiple IoT gateways was conducted on the different hardware with enough capacity.

Because tasks for the data processing on the IoT gateway are not congested, all the transactions in each client can be covered in the device with the specifications as shown in Table 4.2. Moreover, the computational processing is performed not in the client but in blockchain. The client only submits transactions and receive the results. There is no difference in using other hardware only for the simulation.

Hyperledger Explorer Installation

Since blockchain is a sort of network system, transactions and operational processes are not visible. There are no graphic user interfaces, no access via web browsers and no client application software. For the better interfaces with users, additional user applications should be installed. In terms of monitoring the operation, Hyperledger project supports a dashboard called Hyperledger Explorer [47].

Hyperledger Explorer provides web-based access to browse activities and status of blockchain. The network configuration, block and transaction information, and other relevant information can be viewed with this user-friendly tool.

The proposed system is deployed as shown in Figure 4.3 and 4.4 by Hyperledger Explorer. Through this web-based tool, it can be seen that 884 blocks were created by the test and 7,699 transactions were involved in the blocks. Each block listed in the dashboard includes its own hash and the previous block's hash as it is a blockchain.

4.1.3 Evaluation Methods

Through both functional and non-functional tests, the proposed architecture was evaluated. Based on the test scenarios, each functionality and the performance were examined, step by step.

The evaluation of blockchain, the core part of the proposed architecture, was facilitated by Hyperledger Caliper, whereas the entire transaction flow was tested by developed Node.js scripts.

In the architecture, the clients transmit sensor data to blockchain to process. According to the condition, it can be stored or discarded on the blockchain, and then the result is sent back to the client which is an IoT gateway. In this transaction flow, the total response time will be measured with workload simulation. However, the most popular performance testing tools, such as Apache JMeter [48] by Apache Software Foundation and Hyperledger Caliper, are not appropriate to be applied to the blockchain system where the total latency time between external clients and blockchain should be evaluated. Because JMeter is mainly used for HTTP request and response measurements, it is unable to measure the latency in between clients and blockchain. Likewise, Caliper is only for blockchain side. Therefore, Caliper is used for blockchain and the developed script is used for the measurement of the latency between external

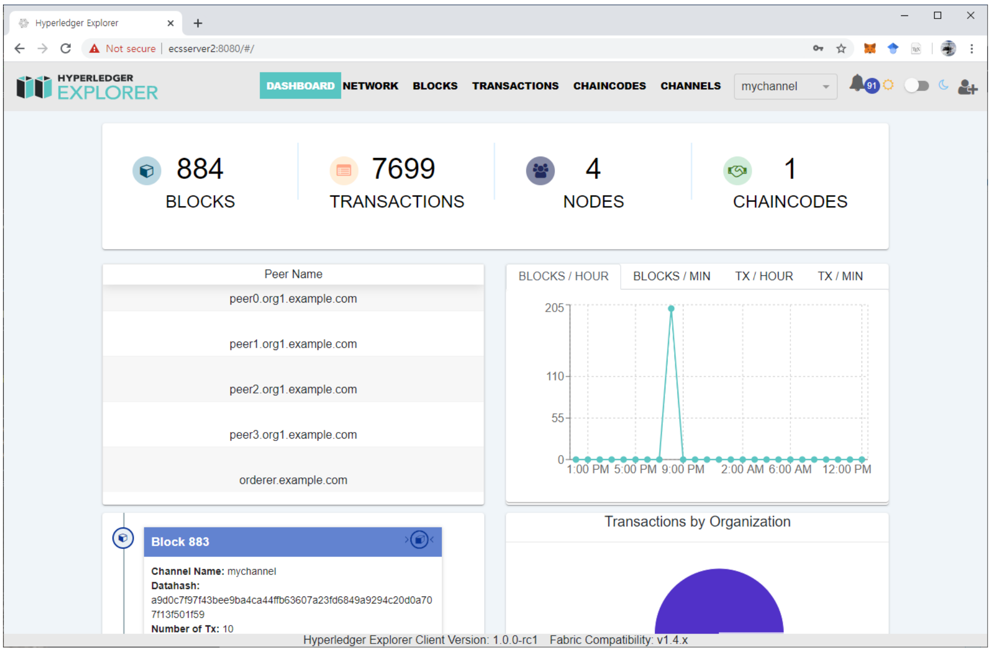

Figure 4.3: Hyperledger deployment monitor - Dashboard

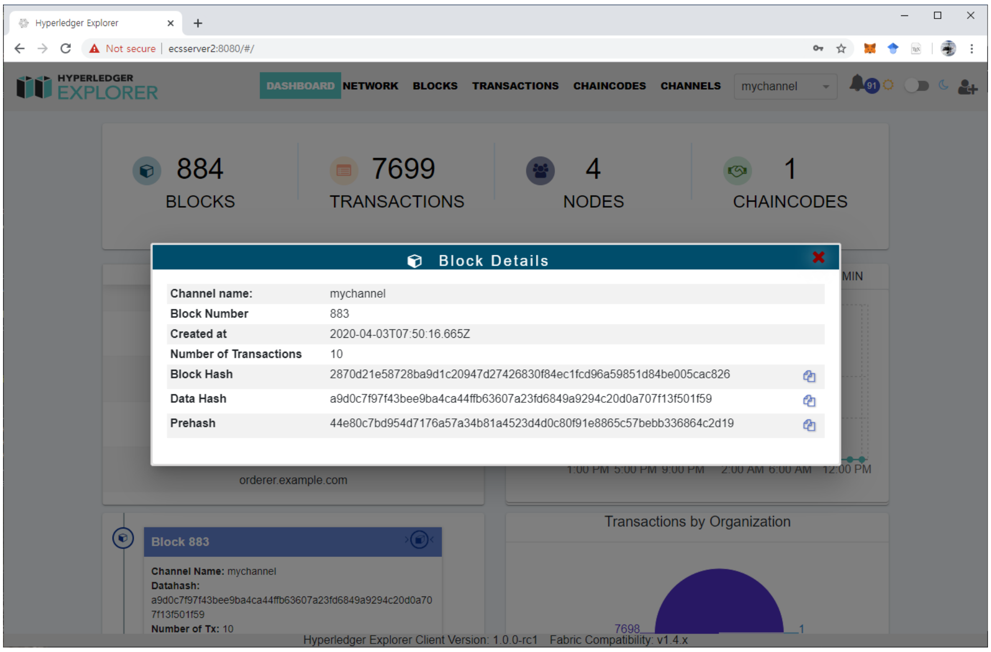

Figure 4.4: Hyperledger deployment monitor - Block

IoT clients and blockchain.

In order to accelerate the performance, parallel processing is normally used. In Node.js, asynchronous functions are used to achieve this. If a single transaction is submitted, it will be easier to check the latency. However, in case of multiple requests simulations, the accuracy of the measurement is related to Node.js asynchronous function mechanism, the specification of which is not defined formally [49], although some performance evaluation tools might still use the function.

Client applications written in Javascript language basically run a single thread processing, so that parallel processing by multiple asynchronous functions may produce delay or waiting time in sequential load of each thread. This latency can be comparable to the delay in the job queue in the server side. Overhead due to the simultaneous requests from clients incurs delay in the server side.

However, the latency caused by multi-threading in a client node should be differentiated rigorously from the one in job queue in a server node in terms of performance measurement. In the job queue, the mechanism, First-In, First-Out (FIFO) is generally applied. When a transaction round trip time in a server is evaluated, it just includes the delay in processing.

In contrast, when the execution time for each transaction that is invoked by the asynchronous function of Javascript in a client node is measured, each transaction has different latency which shows an uneven pattern. It is because a transaction that is submitted earlier is not guaranteed to be processed by priority under the mechanism of asynchronous functions in parallel. The execution of transactions is not sequential. Each response time is affected by the system state at the very moment when the transaction is submitted. Therefore, the measurement by simulation in the client node needs to be supplemented for the accuracy.

To be more specific, the Async function in Javascript operates identically to an object, Promise, so that when an application proposes a transaction, the function is promised to get a success result unless it receives an error. This does not ensure the exact measurement of the response time when the function has received the result. In other words, once the Promise object is declared, the transaction has been already proposed and then the application will handle the result later whether it is a success or not.

The response time for the proposed transaction is affected most by the network configuration. If multiple IoT gateways send the request simultaneously, each gateway only processes a single request on itself, so that the latency in network traffic can be ignored in each gateway. However, in case of the simulation in one client node producing multiple parallel transaction requests, network congestion will arise in the client node, which accordingly effects the response time significantly. Therefore, it is more worthwhile to evaluate the latency in blockchain with connected peers aside from clients.

In blockchain, more important factors of the performance are transaction verification on blockchain rather than the interface. Therefore, the evaluation is conducted in this regard.

4.2 Manufacturing Automation Use Case

Based on the proposed architecture, the design concept is applied to a manufacturing automation process that involves IoT sensors, data transmission gateways, and data integration on blockchain. This use case is an implementation of the design presented in Chapter 3, such that more technical matters are covered in this section.

4.2.1 State Database on Blockchain

First of all, the NoSQL ledger database document structure is defined on blockchain as shown in Tables 4.4, 4.5, and 4.6.

Device information and sensed data can be all included in one record with the composite key consisting of gateway ID, actuator ID and timestamp, for instance, but this is highly inefficient as each attribute has different characteristics. Device information is rarely changed whereas sensed data are generated frequently. Subsequently, they may as well be separated in different documents.

State Database : IoT devices

For the device information document, IoT gateways and actuators are involved. Each IoT gateway S_i of a group of gateways maps its corresponding actuator A_j as in the subset relation expression 4.1, where i and j are indexes. The mapping between gateways and actuators is many-to-one relationship. Each actuator is controlled by the sensed data that at least one or more gateways transmit. Each gateway sends the data towards only one actuator. The limited number of actuators for the facility management are deployed in a production line in manufacturing. Compared with actuators, relatively far more IoT gateways are installed to gather environmental data accurately. IoT gateways are small and cheap.

$$\begin{aligned}
&For\ each\ S_i \in \{S_1, ..., S_x\}\ (i, x \in Natural\ Number,\ \mathbb{N}),\\
&\exists\ j,\ k,\ ...\ \in \mathbb{N}\ and\ j < k,\ ...\ \leq\ x,\\
&\forall\ S_i\ (1 \leq i \leq j) \quad \longmapsto \quad A_j,\\
&\forall\ S_i\ (j+1 \leq i \leq k) \quad \longmapsto \quad A_k,\\
&\cdots
\end{aligned} \tag{4.1}$$

Each IoT gateway has its own hardware serial number and installation location information, so that in case of hardware failure, the device can be identified easily. Each IoT gateway ID is unique, such that the gateway ID is the key in the document.

The corresponding actuator can appear with its own ID and the current operational status. However, if so, when all the input data with the same sensed value but from different gateways are going to update the same

Table 4.4: State database document structure [IoT devices]

Attribute	Type	Description
IoT gateway ID	string	Unique ID for IoT gateway
IoT gateway location	string	Sector No. where IoT gateway attached
IoT gateway S/N	string	Serial No. of IoT gateway
Actuator ID	string	Unique ID for actuator

Table 4.5: State database document structure [Actuator status]

Attribute	Type	Description
Actuator ID	string	Unique ID for actuator
Actuator status	string	Current actuator status

actuator's status, the status is just updated to the same value over and over at all times.

For example, suppose that three different gateways S_1, S_2 and S_3 are in the same group mapped onto one actuator A_{10} as described in expression 4.1. After the gateway S_1 updates A_{10}'s status, if the gateway S_2 checks the status to try to update it, the status is still shown not to be updated as shown in Figure 4.5. It is because two different keys for S_1 and S_2 have independent records. When a value for S_1 is updated, it does not affect the value for S_2 as they are discrete records. However, once the status has been updated, it should not be updated in the predefined time slot as it is redundant.

This repeated update occurs because actuator ID is not a key but a value. Therefore, the status of the actuator should be separated from the device registration information document. Consequently, the document structures is defined as in Table 4.4 and 4.5 respectively.

Thus, the same sensed data from different gateways for the same actu-

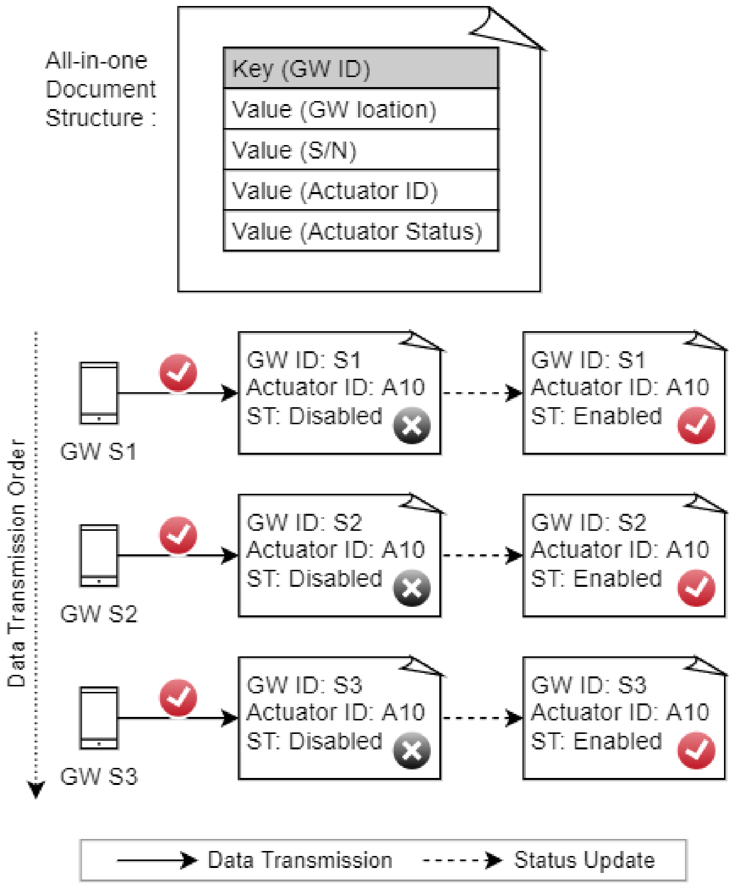

Figure 4.5: Repeated status update case by all-in-one document structure

ator do not update the status repeatedly as can be seen in Figure 4.6.

State Database : IoT sensor data

Sensor data structure is defined as in Table 4.6.

The sensed data are for the actuator control decision, and the data hash is for the data integrity check. Since in the architecture all the sensed data are uploaded to the IPFS, the input data carry its address in IPFS, for the data audit in the future.

As this format, input data are transmitted to blockchain. If the update succeeds on the blockchain, this document is stored in the ledger database.

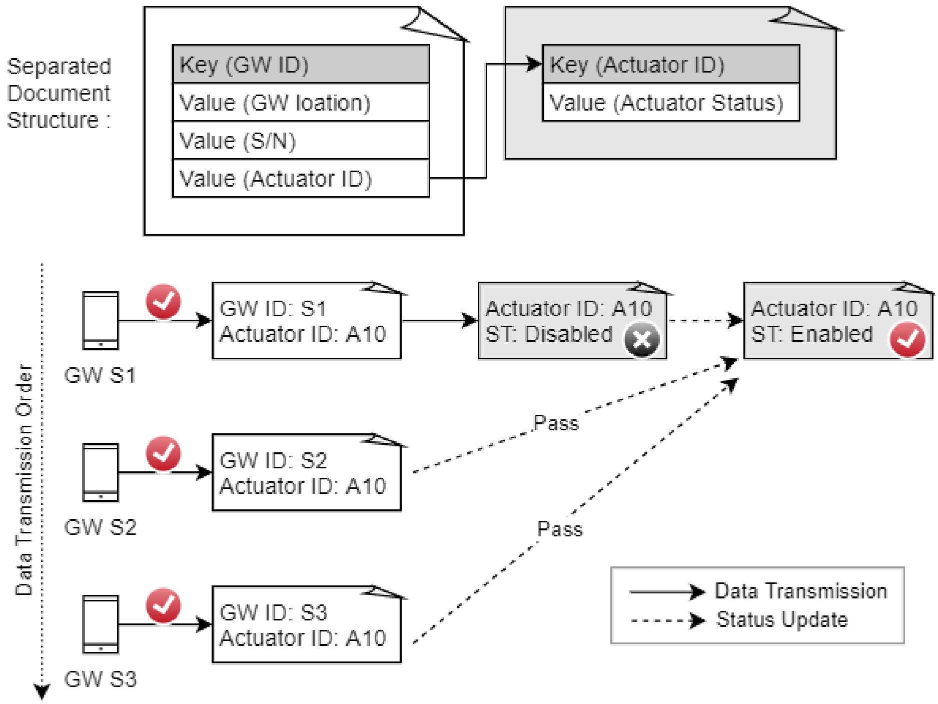

Figure 4.6: Single status update case by separated document structure

IoT data volume is huge, so that only those data which make changes to the system are preserved on blockchain for further check later when needed. Although not defined in the structure, internal version management for each transaction is activated in the ledger database, so that all the records can be identified.

4.2.2 Sensor Data Processing on Blockchain

Once the input data have been transmitted to blockchain from IoT gateways, it is processed by the chaincode already instantiated on blockchain.

The transmitted IoT sensor data should go through several verifications for processing on the blockchain. Above all, IoT data integrity is validated as shown in Algorithm 1. Sensor data can be corrupted during the transmission, so that the integrity check is the first step to process sen-

Table 4.6: State database document structure [IoT sensor data]

Attribute	Type	Description
IoT gateway ID	string	Unique ID for IoT gateway
Actuator ID	string	Unique ID for actuator
Sensed timestamp	string	Timestamp when sensed
Sensed data	string	Sensed environmental data
Data hash	string	Hash value of sensed data
IPFS data address	string	Content addr. where IoT data is uploaded

sor data. Since the IoT gateway communicates with blockchain in a secure way using the digital signature and the transport layer security (TLS) protocol, to check the data integrity is focused rather than to detect malicious manipulation.

The hash function is used for this check. The IoT gateway calculates the hash value for the sensed data from an IoT sensor, appends the hash value to the tail of the data, and then sends the data to blockchain. On blockchain, the chaincode takes the integrity check by calculating the hash value again using sensed data part in the packet and compares the calculated value with the hash value included in the data.

Algorithm 1: IoT sensor data integrity check

input : {IoT Gateway ID, ActuatorID, Sensed Timestamp, Sensed Data, Data Hash H_{data}}
output: {Return Success or Exit}

Calculate Hash H_{cal} *based on Input data*;
if $H_{cal} == H_{data}$ **then**
  return Success;
else
  exit;
end

After the integrity check, the input data itself are regarded to have no fault in terms of technical aspect. Then, the delay check is performed as in Algorithm 2.

Algorithm 2: IoT sensor data delay check

---

input : {Sensed Timestamp TS_{sense}}
output: {Return Success or Exit}

Retrieve the current system time T_{sys}*;*
Convert T_{sys} *to timestamp* TS_{curr} *in calculated seconds;*
if *(*TS_{curr} *-* TS_{sense}*)* $\leq$ *allowed time threshold duration* **then**
    return Success;
else
    exit ;
end

The delay allowance is defined in the chaincode. This allowance is set based on the business requirements in practice. The timestamp of the input data, which is the time of sensing, is compared with the current system when the data are processed. If the gap is beyond the predefined time threshold, the input data are not processed. Such data are considered to be invalid for the actuator's action.

Among a number of data from IoT gateways, one specific data may still have a significant effect even though it is delayed. Nonetheless, there are multiple other sensors and gateways in the vicinity in the proposed architecture, so that the operation can rely on the signal from the majority. Unless either parts of the network system or devices are down, the delayed data itself should be disregarded since the information it carries is outdated.

Finally, for the state database update, the sensed data are evaluated as in Algorithm 3. After the state is updated successfully, the relevant action is taken by the actuator corresponding to the gateway in the input data.

Algorithm 3: IoT actuator status update

input : {IoT Gateway ID, ActuatorID A_{id}, Sensed Data S_{data} }
output: {New Status, Event (Payload), Return Pass or Exit}

Retrieve current actuator status A_{status} *of* A_{id}*;*
switch do
    case $S_{data} >$ *upper threshold &* $A_{status} ==$ *enabled* **do**
        return
    end
    case $S_{data} >$ *upper threshold &* $A_{status} ==$ *disabled* **do**
        set A_{status} enabled & emit Event
    end
    case $S_{data} <$ *lower threshold &* $A_{status} ==$ *enabled* **do**
        set A_{status} disabled & emit Event
    end
    case $S_{data} <$ *lower threshold &* $A_{status} ==$ *disabled* **do**
        return
    end
    otherwise do
        return
    end
end

The current actuator status is retrieved from the ledger database, and sensed data are compared with the predefined threshold for the actuator bootstrap in the chaincode.

If the actuator is enabled and it needs to be disabled by another sensed data in the future, the chaincode iterates the process as defined in the algorithm and vice versa. As a result, the status of the actuator switches based on the input data.

To be precise, the chaincode updates the state database, emits an event with payloads, and stores the transaction on blockchain at the same time.

The payloads contain the signal for the actuator's action. Upon receiving the emission in the application process that is listening for it all the time, the application invokes the actuator's activity.

4.2.3 Sensor Data Processing on IoT Gateway

Sensed data are transmitted from sensors to gateways, and the gateways will determine if the data are preserved or not dependent upon the regulations in the organisations in practice. The compliance with data preservation for auditing is assumed in the proposed architecture, so that if it is unnecessary in practice, the preservation process can be eliminated from the architecture without compromising its function.

Muralidharan et. el. [50] introduced IPFS for IoT sensors' peer-to-peer network. They configured a private IPFS cluster overlay network on the IoT sensor network to prove their concept, but in that case, the idea does not work well in the resource constrained IoT environments as each IoT node in their system was supposed to store distributed data in its local storage. Given a huge volume of IoT sensor data generated, this idea is not capable of processing data. In the proposed architecture, instead, the real public IPFS network is used to store the IoT data, regarding the storage capacity of IoT devices. In most cases, hard disks are not attached to IoT devices, so that IoT devices should rely on their cache or SD card for the data storage whether it is temporary or not, which is volatile and fragile.

In a bid to reduce the data traffic on blockchain effectively, sensed data that are beyond predefined criteria should be filtered in IoT gateway. However, if the threshold needs to be changed frequently, the threshold set in the gateways should cover a wide range of allowances in order to offset the frequent policy changes. It is because the number of deployed gateways will be large, so such maintenance costs a lot in terms of labour and may cause inconsistency by any chance. Therefore, the threshold in IoT gateways should be a bit insensitive range inclusive. Instead, the accurate threshold for each operation is set on the chaincode in blockchain.

With regard to this scheme, IoT gateways will forward sensed data that beyond their own threshold to blockchain, and the rest of data transmitted from sensors will be sent to IPFS.

4.3 Comparison with Ethereum

Although Bitcoin introduced the concept of blockchain for the first time, blockchain was only used as an underlying technology for cryptocurrency. It was Ethereum that extended blockchain to many other applications. Basically, Bitcoin is a platform for digital money transactions, while Ethereum is a "turing complete" platform [9] as it has versatile extensions with the smart contract. Therefore, in this thesis, Ethereum is compared with the proposed architecture.

Since Ethereum is based on the public blockchain and permissionless processing, it may not suitable for enterprise in terms of data privacy. Moreover, it is based on anonymous consensus, which increases latency in validation of a transaction dramatically and is known to process 15 transactions per second. Therefore, in terms of performance, the public blockchain cannot be comparable to the private blockchain fundamentally.

Technically, the consensus algorithm and the membership management are two main differences between public and private blockchain as both blockchains are distinguished by the participation of the network. The public blockchain is normally permissionless, which means it does not need to manage the participants whereas in the private blockchain, as it is private, the management of membership is one of the foremost functionalities. On the other hand, participants to the public blockchain are anonymous, so trust really matters among trustless participants. This trustworthiness can be achieved by various consensus algorithms in public blockchains, although this procedure causes significant degradation in performance.

Subsequently, the first thing to do to operate the blockchain network is that the membership management should be set by using Hyperledger membership service provider (MSP) and client applications should use credentials created by a certificate authority system to access blockchain and smart contracts.

Even though consensus mechanism can be regarded relatively with lower priority than in the public blockchain, the private blockchain also has the mechanism. However, since the design architecture aims at IoT domain and high priority is put in autonomous operations, the endorsement policy is set to at least one endorser's agreement which is an 'OR' condition, rather than all the agreement from all endorsers.

As a matter of fact, because of the latency and the costs to submit transactions on Ethereum, experiments in relation to Ethereum shall be performed on an Ethereum test network, but the private blockchain can be provided with its own private network, which is one of conveniences.

4.4 Summary of the Chapter

This chapter presented the validation environment based on the proposed architecture in Chapter 3.

The implementation of PoC model system on a Linux machine was described with the following proposed algorithms.

- IoT data integrity check
- IoT data delay check
- State database update

The data structure designs for the ledge database were declared by using a use case.

It is checked that the system is running up in the Hyperledger Explorer, the blockchain monitoring tool provided by Linux Foundation.

The differentiations and comparisons with Ethereum that has a number of use cases were additionally explained from the practical point of view.

General limitations of simulation methods in application stress and load tests were addressed in this chapter. Taken the limitations into account, the experiments on the simulation environments are proceeded as in the following chapter.

Chapter 5

Experimental Results

Though blockchain has been adopted in many industries for the past few years, most adoptions have been struggling to improve performance and scalability in many ways, because some have inherent drawbacks such as high latency while others have high configuration complexity. The struggles are mostly caused by the fact that the technology is immature.

The peer's role in blockchain is important as it is based on decentralisation. In decentralised systems, all the participants are supposed to process transactions, whereas in centralised systems, the central authority controls the process. As other distributed systems do, the more computing resources in blockchain are deployed, the better the performance becomes. However, in blockchain, there is a consensus procedure to process transactions without the central authority and more resources do not necessarily guarantee the better performance. It is because the total transaction execution time is related to the size of blockchain, the number of participants in the blockchain. If the number of participants grows, the latency for the consensus will increase proportionally. The resource expansion by a scale-up will accelerate the processing in a peer node, but adding peers by a scale-out in a blockchain deteriorates the performance.

In IoT environments, hundreds or thousands of devices are connected to the network, and so the peer nodes should have enough capability to handle as many devices as possible. Otherwise, the alternative is to deploy more peer nodes on the network. However, the more IoT devices a node embraces, the higher the risk of failure in that group of devices becomes. On the other hand, adding more peer nodes cause the performance problem as mentioned above. Therefore, it is crucial to balance the workload distribution. Focusing on the adjustment, the experiments were performed in a more qualitative approach and analyse the results extensively in this chapter.

The overall experimental results both of functional and non-functional tests are reviewed. The functional tests focus on the verification of the proposed architecture, whereas the non-functional tests mainly deal with the performance and the stability. In order to check the performance, the virtual clients are simulated and the experiments are repeated by increasing the load. Then, the results are compared with other works to examine that the proposed architecture produces a prominent enhancement.

Based on the proposed concept, it is shown how the architecture maintains the data integrity in IoT data processing on blockchain. In the last part of this chapter, further additional issues that occurred in the non-functional tests are discussed.

5.1 Test Methods

The test methods are classified into functional and non-functional testing.

5.1.1 Functional Test

Generally, the functional tests are performed by developers and end-users in the case of enterprise application developments. Thus, in order for the experiments to be more objective, each of the functional test is defined as in Table 5.1 referring to generally accepted usages [51]. Then, the test scenarios are listed up in detail as in Appendix B.1.

Table 5.1: Functional test type

Test Type	Description
Unit	required parameters and return value check
Smoke	build verification
Sanity	major and vital functionalities working as intended
Regression	code change has not adversely affected existing features
Integration	multiple functional modules working together
Usability	for production usage by users (similar to UAT)

The result for each criterion is examined step by step. Firstly, the basic functionalities of blockchain including start-up and shut-down were checked. Then, the algorithm was verified, which corresponds to the user acceptance test (UAT) in practice. By the Usability Test, it was identified that data integrity was preserved with the proposed architecture.

Data Integrity Check

Above all, data that do not follow the declared data format should not be processed, so that faulty input data were created to check whether the smart contract filters it out. Only if each task returns an error in this scenario, the input data turn out to be well integrated and processed on blockchain. In this way, it is inferred that the data integrity is guaranteed.

The below input data fields were manipulated in several ways.

- Value of one or two arguments
- Hash value
- Argument type (string vs integer)
- Using unregistered device ID

When any input arguments other than the hash value of the input data was manipulated, the smart contract returned error. Likewise, when the hash value was changed, an error is asserted. The check for the data integrity is performed by Algorithm 1 in Section 4.2.

In particular, all the IoT gateways were registered in the state database, so that when an unregistered IoT gateway ID is submitted, the smart contract returns an error. This also shows that data from an unauthorised device cannot submit a transaction.

Transmission Delay Check

The delayed transmission cases were tested through Algorithm 2 in Section 4.2.

The timestamp value in the input data was manipulated in two ways. One was conducted by using arbitrary values, and the other was by timestamps beyond pre-defined time gap threshold. The former is to check the processing of the timestamp data format in the smart contract and the latter is to check how the smart contract deals with the delayed data.

In both cases, transactions were stopped with an error, which means a success. The time gap threshold was set in seconds in the smart contract, and the simulated data with timestamp over this threshold were not processed.

Thus, the delayed data processing based on the design was ruled out and the manipulation of the input data was checked, which eventually reinforces the data integrity.

State Update Check

Next, the evaluation was conducted with the test data that would update the status of an actuator in the ledger database. The update is processed based on Algorithm 3 in Section 4.2.

A query to the current state of the ledger found the status of an actuator. Then, the environmental data that would change the status was generated. The data should not be within the predefined threshold in order to change the status. When the update transaction with the generated data is submitted by the client application, the ledger was updated as expected. When the ledger is updated successfully, the result with the transaction ID was returned to the client application and an event was emitted to another client application which is the actuator controller in the architecture. Upon receiving the event with a payload defined in the smart contract, it is deemed that the actuator took a relevant action.

For the above evaluation, when the status of the actuator was enabled, the data that could disable the status was generated. It is assumed that an air conditioner existed as an actuator and the generated temperature was an input data. The submission of the transaction with the data was simulated by a developed client application. The action in the actuator was regarded to be taken properly when another client application for the actuator received the event payload successfully.

When the air conditioner was being turned on, the temperature data over the upper limit threshold to turn off the air conditioner were generated. During this test, it was also checked whether the status remained unchanged when the temperature data did not meet the criteria, which was successful.

By this update check, it is identified that the proposed architecture processes IoT sensor data and controls the actuators properly, which shows the autonomy.

5.1.2 Non-Functional Test

The non-functional tests are mainly focused on such aspects as performance and stability independent of business applications. Though the user acceptance is dependent upon Usability Test in the functional tests, system performance has a decisive effect on the user experience. Hence,

Table 5.2: Non-Functional test type

Test Type	Description
Performance	evaluate the overall performance of the system
Stress / Load	validates the system performs as expected under stress
Volume	verify when a large volume of data is involved
Compatibility	evaluates the application is compatible with others
Recovery	verify against HW and SW failures
Failover	verify in case of a system failure
Security	ensure that the application has no loopholes
Scalability	verify the application is capable of increased requests
Localisation	verify the application in different languages
Benchmark	use a base for any new application

the performance test is separated from other non-functional tests and performed it intensively.

As in the functional tests, each type of the non-functional tests is enumerated as Table 5.2. The non-functional tests are based on the general classification [52]. According to the test types, detailed test scenarios are elaborated in Appendix B.2. With the test scenarios, each activity was tested.

Differences in recovery from centralised systems

In enterprise applications, it is critical that data should be preserved when hardware failure occurs and the service continuity should be guaranteed. In many centralised system environments, high availability is mostly implemented by replication and distribution with multiple resources. Data are backed up to different medium.

However, blockchain runs on the distributed nodes and the shared ledger database, hence it is fundamentally more robust than the centralised system. Accordingly, the tests against hardware failure are a bit different, although the aim of recovery tests will be the same.

The differences lie in backup medium, restore process, and role of a node. In the centralised system, which node has a fault is the main concern. If it is the central node, the recovery is also complicated. However, in blockchain, every node is identical in terms of authorisation. Among all the participating nodes, it is distinguished that a node is either an endpoint distributed resource or a process leading anchor. In spite of the differences in roles, the recovery process is not affected by a hardware failure in any node as the role can be switched automatically by its status.

In a blockchain, the recovery of a node does not request system administrators' commitment, as all the data on blockchain can be shared again by other alive nodes when the crashed node is restarted.

Recovery Test

It would be nearly impossible that all the nodes are altogether down at once in most popular public blockchains. In private blockchains, it would be rare as well, but be possible depending on the size of blockchain. This extreme case in private blockchain was tested by Recovery Test.

In Recovery Test, blockchain managed to be restored given that a few essential objects are backed up in a safe condition. The essential objects are such things as:

- Crypto materials
- Channel artifacts
- Peer data: Chaincodes, Ledger database file, Blockchain file
- Orderer data: Configuration file

Failover Test

In centralised systems, the failover can be achieved by taking over the role of the crashed node by another node. The node that takes over can be either in a stand-by mode as a redundant hardware resource or in an active mode in which it provides its own service at normal times and is changed in case of the failure.

In blockchain, a node does not take over other node for the hardware resources. Therefore, the failover was tested in a different way only to guarantee the service continuity by other nodes.

One peer node was stopped and then two nodes was stopped. While peer nodes were stopped, transactions were submitted and it was found that they were processed successfully.

This experiment, however, depends on the endorsement policy. If the endorsement policy is set to 'AND' and the crashed node is an endorser, the transaction is not verified. For this reason, in practice, sufficient numbers of endorsing peers should exist in blockchain.

Thus, through Recovery test and Failover test, it was identified that the data were secured with more resiliency than the centralised systems even when some nodes were crashed.

Scalability Test

For the scalability test, nodes were incrementally added to the blockchain.

Compared with the centralised systems, a blockchain only needs the following tasks in adding a new node.

- Generate credentials for a new peer to be authorised in the blockchain
- Fetch the existing blockchain configuration to a new peer
- Update the existing blockchain configuration
- Submit the update to the blockchain

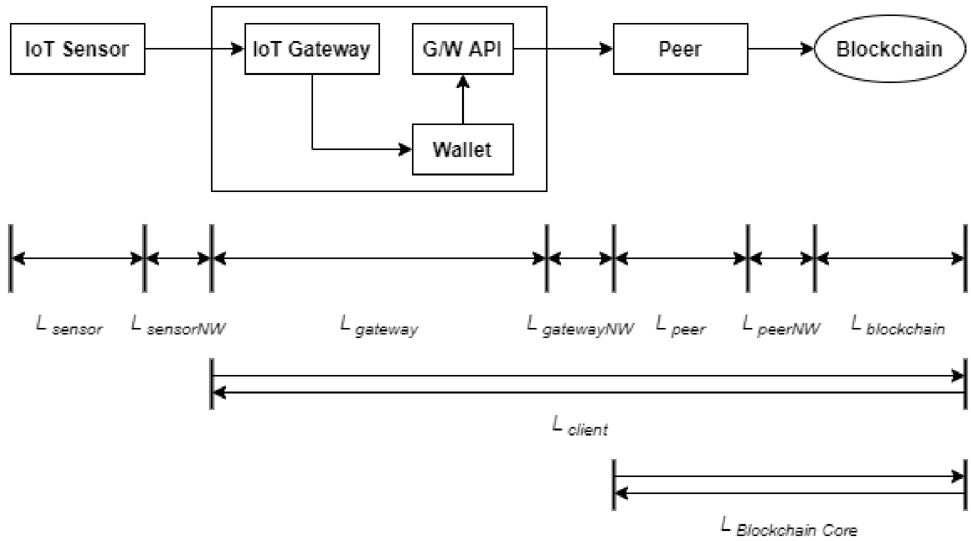

Figure 5.1: Latency measurement in components

- Join the blockchain

The changes due to adding nodes are processed on blockchain as a transaction. Then, the information is distributed among all the participating peer nodes. Thus, it achieves the scalability.

Several peer nodes were successfully added on the blockchain online. Moreover, hardware resource consumptions in each case with different numbers of peer nodes were measured to check its operability. The results of the resource utilisation are discussed in Section 5.5.

Performance Test

As aforementioned, considering the importance of Performance Test, it was conducted intensively in a variety of ways.

In overall, the latency can be measured by each interval as shows in Figure 5.1. The sensing time L_{sensor} is entirely dependent upon the computational resources of the sensor, so that it is not considered in the experiment.

Each latency L_{sensorNW}, $L_{\text{gatewayNW}}$, and L_{peerNW} passing through the physical network section between each node is not taken into consideration either as it completely depends on the network bandwidth and the routing configuration.

The total response time L_{client} is composed of the latency from a client application through blockchain processing and back to the application as shown in Equation 5.1.

$$\begin{aligned} L_{\text{client}} = \ & (L_{\text{gateway}} + L_{\text{gatewayNW}} + L_{\text{peer}} + L_{\text{peerNW}} + L_{\text{blockchain}}) \\ & + (L_{\text{blockchain}} + L_{\text{peerNW}} + L_{\text{peer}} + L_{\text{gatewayNW}} + L_{\text{gateway}}) \end{aligned} \quad (5.1)$$

The latency in blockchain core section $L_{\text{Blockchain Core}}$ is determined as Equation 5.2.

$$\begin{aligned} L_{\text{Blockchain Core}} = \ & (L_{\text{peer}} + L_{\text{peerNW}} + L_{\text{blockchain}}) \\ & + (L_{\text{blockchain}} + L_{\text{peerNW}} + L_{\text{peer}}) \end{aligned} \quad (5.2)$$

Special attention is paid to L_{gateway} and $L_{\text{Blockchain Core}}$ in the experiments as they are affected by the proposed architecture.

In Figure 5.1, the actuator, another client in the whole system landscape, is not depicted. The actuators access blockchain in the same way as IoT gateways do as they are all external clients of blockchain.

The actuator application retrieves locally stored private key to call the blockchain client gateway API to access blockchain. Then, it waits for the event which is emitted from blockchain whenever the ledger update transaction is successfully processed.

Since the actuator is supposed to take immediate action based on the messages from blockchain, an event listening daemon process of the actuator should be constantly run. Otherwise, the actuator is triggered after the event arrives at every time, which causes a significant delay in the processing. Therefore, when the application receives the response from blockchain, it can be regarded that the client in other side, the actuator, has already been given the result, so that it does not need to measure the latency in the actuator.

The results from Performance Test is evaluated and analysed in the rest of this chapter.

5.2 State Database Analysis

In Hyperledger, the current status of all data on blockchain is stored in so-called world state database [53] using NoSQL database while transaction history and their blocks are recorded and stored in the blockchain as files on the file system. For the world state database, the latest long-term support (LTS) Hyperledger release supports CouchDB [54] by default. Prior to the current release of Hyperledger, it was associated with LevelDB [55]. One of the reasons why the strategy has been changed is because CouchDB provides various forms of rich queries due to its indexing functionality. However, as in selection of other applications or tools, the purpose of the usages should be looked into in the first place. In the proposed architecture, rich queries are not certainly needed because the application is in the IoT domain rather than the financial business which involves queries with diverse search conditions to monitor the current status changes in between stakeholders.

Meanwhile, LevelDB is known to outperform CouchDB in performance in case of simple query and update as it does not need to consume computational resources to build the index to be used for rich queries. Therefore, it is intended to use LevelDB instead of CouchDB despite the current coordination with the Hyperledger release. However, before experimenting the evaluation, the performance between the two NoSQL Database was compared carefully so as to be confirmed for the proposed system.

The experimental environment consists of the same conditions as the

Table 5.3: Workload configuration for state DB performance test

Object	Property	Value
clients	type	local
	number	1
rounds	tx number	250
	send rate control type	fixed-rate
	send rate control options (tps)	250

performance test of the proposed system. The same application transactions were also used. The only difference is that the Docker image for Hyperledger peer includes LevelDB, so that containers for state database do not run separately in the case of LevelDB whereas CouchDB runs on a discrete container. Since Docker containers represent each physical machine and the containers for CouchDB run on the same machine in the experiment, the utilisation of hardware resources is not comparable.

In the test, each client proposes a transaction 250 times at a speed of 250 transactions per second by using Hyperledger Caliper as described in Table 5.3. The total executed transaction number is one of the experimental cases for the comparison with other study in the performance test, and the sending transaction frequency is sufficiently large to evaluate the performance per second according to the executed transaction number. This comparison test was performed 50 times repeatedly and the average results are shown in the graphs in this section. The error bars in the graphs convey margins of 10%. The transaction was executed sufficiently to obtain more accurate results and the outcomes from each test round converged within 5% of differences, so that the error allowance is set to 10%.

Figure 5.2 shows the latency comparison when IoT gateways and actuators are being registered on blockchain initially. All the latencies evalu-

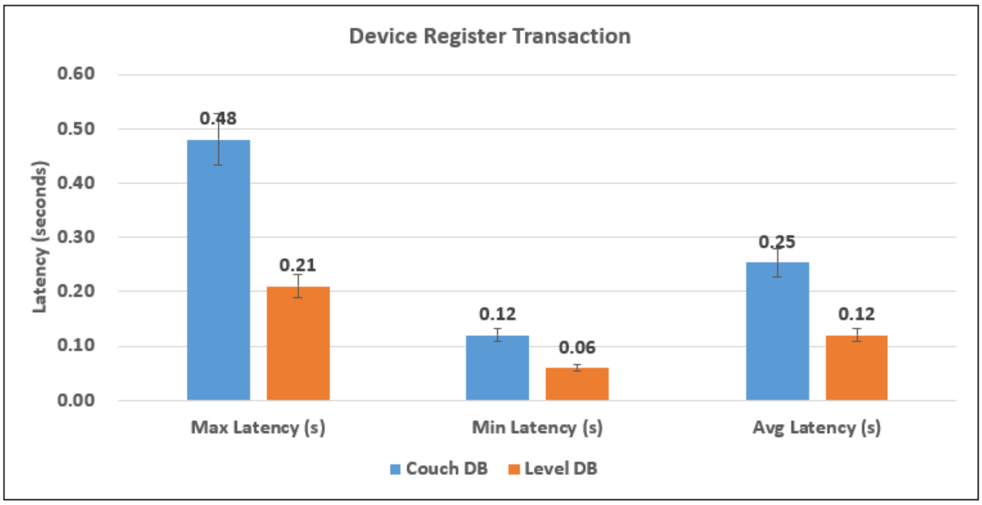

Figure 5.2: Latency comparison in device registration transaction

ated are doubled when CouchDB was used. Maximum latency reaches up to 0.48 seconds in the case of CouchDB, whereas 0.21 seconds in LevelDB. In the case of LevelDB, the difference between maximum and minimum latency is much less than that of CouchDB, the gap of which is 0.36 seconds. The average latency is 0.25 seconds in CouchDB, whereas 0.12 seconds in LevelDB.

In the case of simple query transaction as shown in Figure 5.3, the minimum and average latencies were the same, although there was a slight difference in the maximum latency. Because this test case is for simple query without using complex index, they show no big differences. However, if complex index or pagination function had been applied, the result must have been obviously different.

As shown in Figure 5.2, Figure 5.4 presents the similar results as it is for the status update test. Each latency in Figure 5.4 is slightly lower than each one in Figure 5.2. There are two reasons. One is that the device registration transaction is similar to the insert function as in relational database, which is to create a new record in the database rather than to update a value for an existing key, so that the size of I/O is bigger. The other reason is that

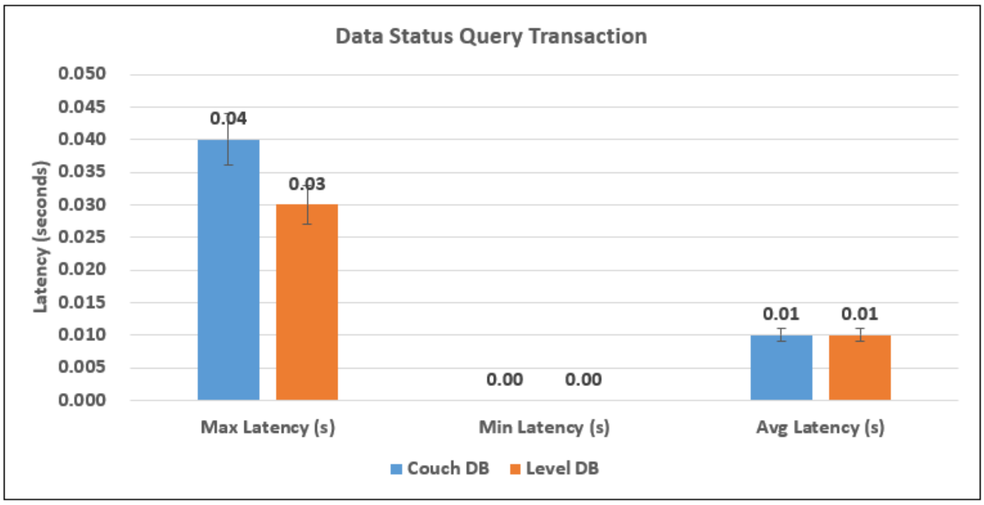

Figure 5.3: Latency comparison in simple query transaction

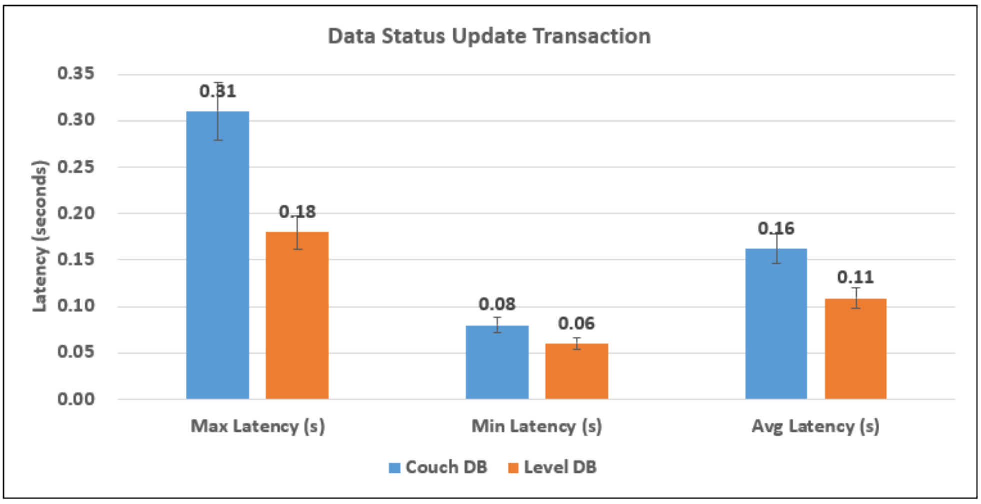

Figure 5.4: Latency comparison in data update transaction

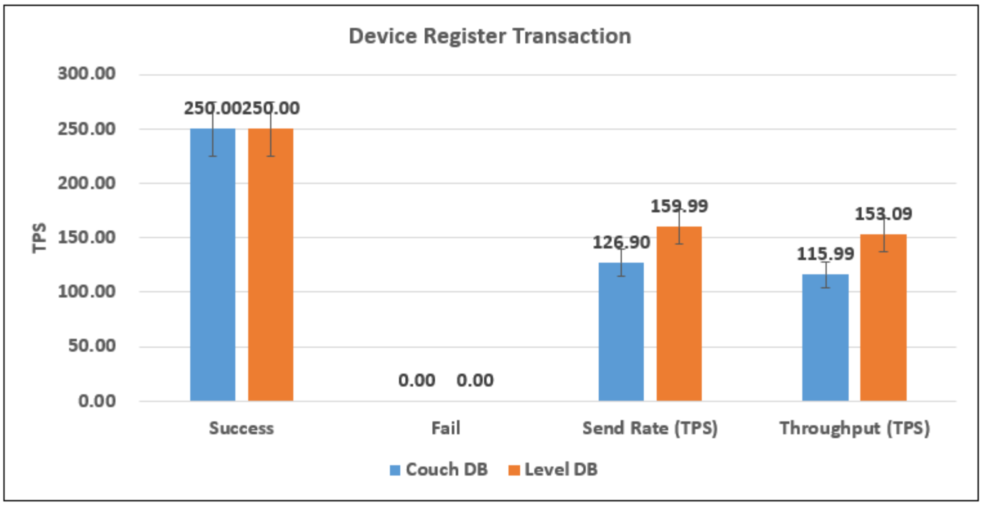

Figure 5.5: TPS comparison in device registration transaction

the update occurs only when all the criteria defined in the chaincode are satisfied with the input data. Subsequently, such transactions exit without database I/O during the process, which eventually caused lower latency.

Next, transaction success ratio and throughput per second are analysed as shown in Figure 5.5. Since the device registration transaction only creates a new record in the database, there is no failure. However, the result shows there are long delays. When 250 transactions per second were submitted, only about 160 transactions in the case of LevelDB were successfully sent in effect. The remainder just waited for their turn while the 160 transactions were being processed. Among the 160 transactions, 153 transactions were processed completely. The sending rate and the throughput are much less in the case of CouchDB than LevelDB, by 33 and 37 transactions per second respectively.

In contrast, the simple query simulation produced the same performance both in the case of CouchDB and LevelDB as shown in Figure 5.6. The sending rate and the throughput were also the same as the intended submission rate as the simple query transaction simply evaluates the data in the ledger without any lock.

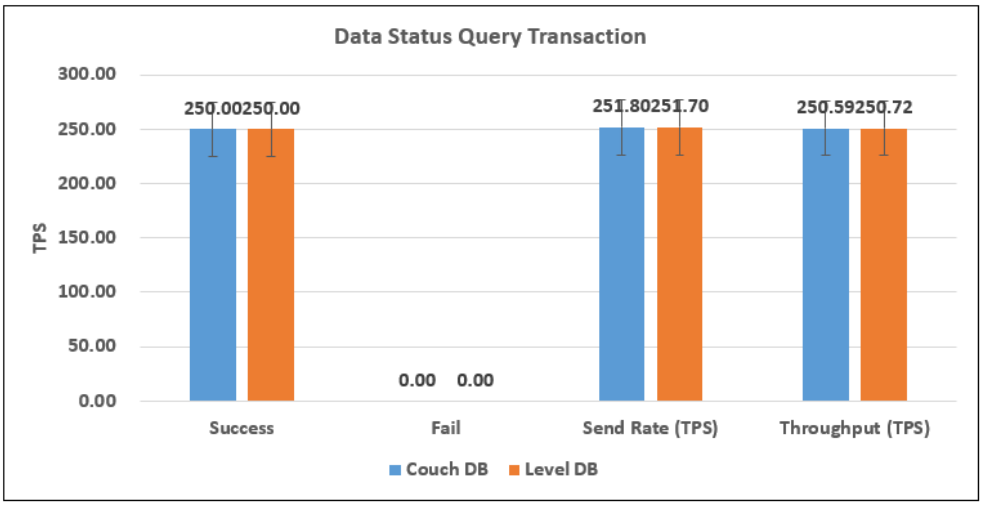

Figure 5.6: TPS comparison in simple query transaction

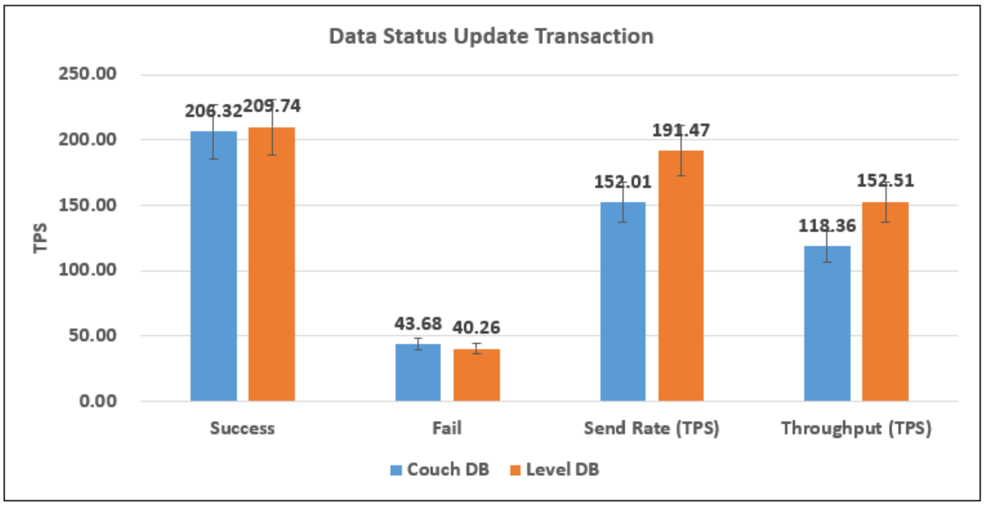

Figure 5.7: TPS comparison in data update transaction

In the data update simulation as shown in Figure 5.7, the trend is analogous to the device registration simulation. The difference is that in this case there are failed transactions because if transactions try to update the same key simultaneously, they collide and only one succeeds. The key selection is arbitrary as this is an automated test. In addition, the chaincode in the simulation checks the appropriateness of the update based on the predefined conditions. Because there are failures in the update processing, the sending rate is better than the one of the device registration simulation. The failed transactions promptly yield for the next submission. However, the number of transactions that have been processed completely in a second shows the same as the device registration, which is related to the performance of the proposed blockchain.

5.3 Transaction Processing Performance

For the performance test, latency in client transaction process including request and response was measured by a physically remote client through blockchain network. The client node and blockchain node are located in different buildings and connected by 1000Base-T.

5.3.1 Data Query Transaction

Figure 5.8 shows the performance test for the data query transaction. Each load was given by 50, 100, 150, 200, and 250 concurrent clients per second made by Node.js script and repeated 50 times. As the experiments in Section 5.2 showed latency at 250 clients, fewer than 250 clients submitted for the performance test. The scalability was chosen in increments of small, medium and large loads by 50 clients.

In general, latency grows slightly linearly as the numbers of concurrent clients increase. All the average latencies are stable and acceptable below 0.3 seconds when IoT actuator operation is concerned, but there are several

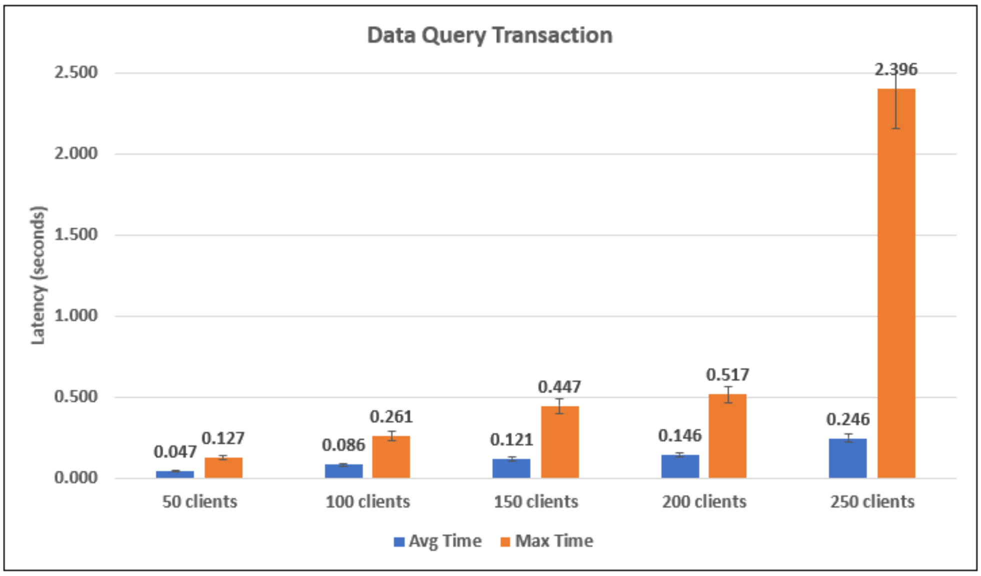

Figure 5.8: Data query transaction latency

spikes that contribute to the anomalous maximum value. The maximum latency soared up to nearly 2.4 seconds in case of 250 clients load. The increase of the average and maximum latencies is gradual when the load is given by up to 200 clients, but with over 200 clients, the maximum latency, in particular, shows the spikes a few times.

Because the load was created in the single client node, the simulation has gone through network congestion in outbound and inbound traffic. The latency was measured in the client node from the start of a transaction till the receipt of the result. In practice, the request is submitted by different IoT gateways, so the network congestion in the client side rarely occurs. Moreover, closer inspection of the latency shows that all transactions longer than 2 seconds occupy merely 0.05% of the total execution in 250 client case, which can be neglected. However, the causes of the spikes will be discussed more in Section 5.6.

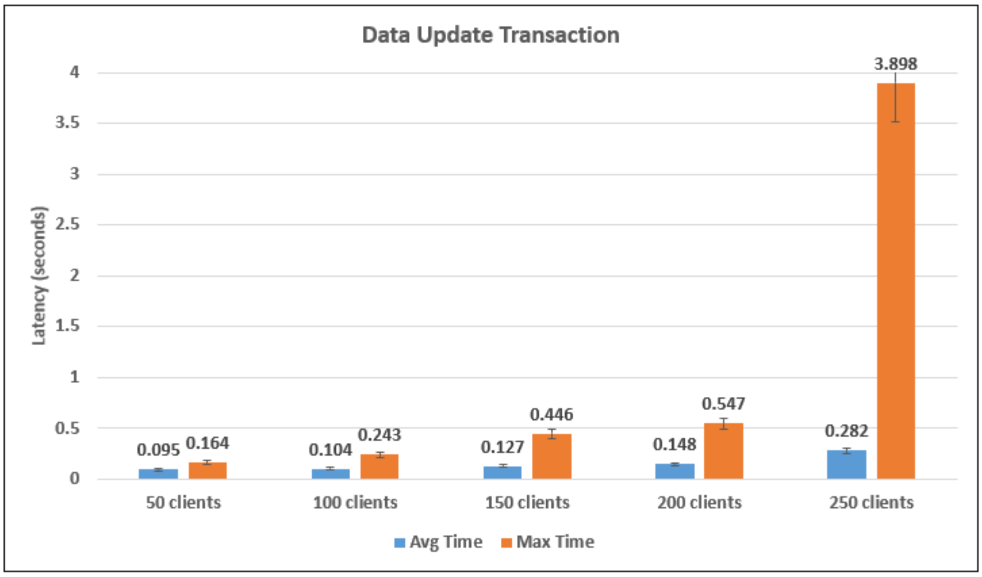

Figure 5.9: Data update transaction latency

5.3.2 Data Update Transaction

As shown in Figure 5.9, the results of update transaction test are similar to the query transaction test in terms of the increasing latency trend. In case of average latency, the gaps between update and query transaction are just below 0.05 seconds for each case. The spikes also appear in 250 client case. The linear increase pattern is caused by the contention as the simulated clients increase. The approximate value of latencies between update and query transaction test has two noteworthy reasons.

Firstly, the similarity is caused by the processing logic in the smart contract and the mechanism of the multi-version concurrency control (MVCC) in database management. Hyperledger Fabric basically adopts the non-blocking algorithm to avoid the ledger MVCC collisions, which makes each thread lock-free and wait-free [53]. Subsequently, when the load test is performed by using concurrent async functions in the test script written by Node.js in parallel, Hyperledger returns the avoidance of the update transaction without update of data in the state database if the dirty read occurs. This MVCC collision is inevitably prompted in the load test as it uses the simulation method with limited number of artificial data and requests. Meanwhile, this mechanism prevents the transaction from invoking database I/O, which will reduce the response time, as the transaction is only processed by the smart contract.

Secondly, because of the transaction handling logic in the smart contract, some of the requests are discarded based on sensed data associated with the update conditions, which also means they do not access the state database likewise. In the proposed architecture, the smart contract decides whether the transaction should be processed further to the state database or not.

Therefore, the differences in latency between update and query transactions are somewhat dependent upon case-by-case, so that latencies of update eventually turned out to be close to figures of query in the simulation. This can be another good demonstration for the proposed architecture as the logic functions well.

5.4 Comparative Evaluation

This section presents two kinds of comparative evaluation. One is the performance analysis by interval and the other is the performance comparison with other work.

The former analyses the performance measurements by section based on Figure 5.1. This analysis enables the accurate measurement of the latency in each section, which helps to find a bottleneck in a transaction flow pipeline.

The latter compares the performance results based on the proposed architecture with other work. In order to judge the results objectively, the comparison is necessary.

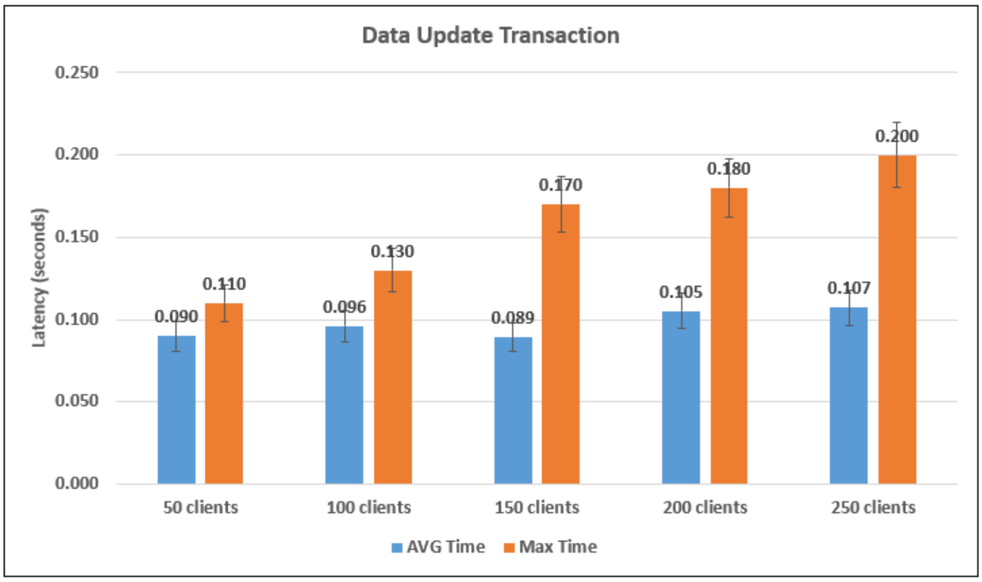

Figure 5.10: Update transaction latency only in the blockchain segment

5.4.1 Comparative Analysis by Interval

$L_{\text{Blockchain Core}}$, latency in transaction processing in blockchain was measured as depicted in Figure 5.1, and then compared with the total round-trip processing time L_{client} from the client node. This enables the measurement of the latency $L_{\text{GatewwayNW}}$ which is highly variable depending on the real environments. This approach is quite significant as the latency of each interval can be analysed respectively. Subsequently, the hot-spot can be easily found, and based on the analysis, fine-grained reconfiguration can be achieved. The data update transaction with from 50 to 250 concurrent application processes was simulated by 50 processes increase, and used Hyperledger Caliper to evaluate the latency in blockchain.

The results are presented as Figure 5.10 for each simulation with different number of clients. The average latencies for each case are around 0.1 seconds with a slight margin, whereas the maximum latencies grow as the client number increases. As the number of concurrent transactions mounts, the possibility of contention naturally increases, so does the max-

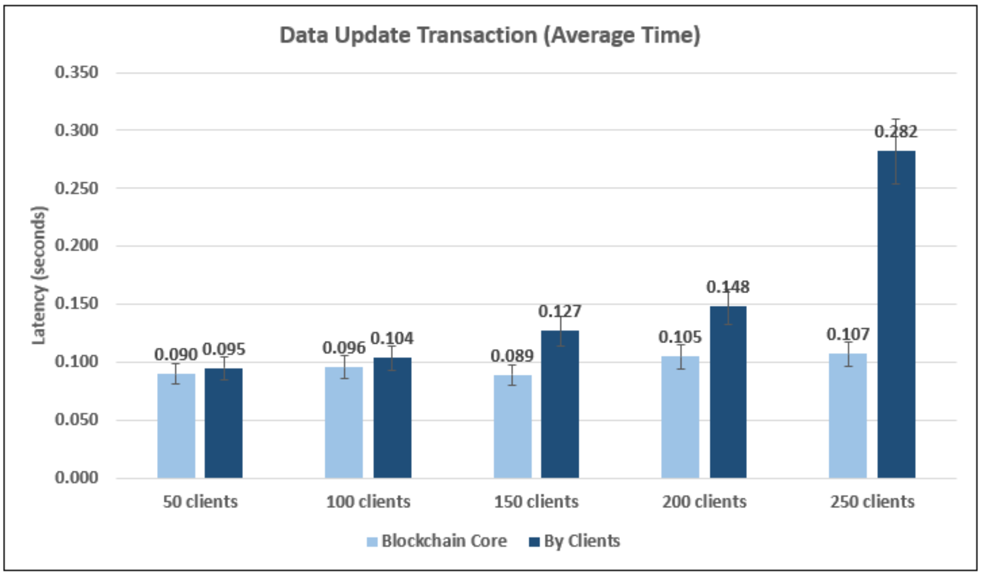

Figure 5.11: Average update transaction latency comparison by interval

imum latency grows. However, the transactions that have long latency are still few, such that they have little effect on the average calculation.

Figure 5.11 compares $L_{\text{Blockchain Core}}$ and L_{client} as shown in Figure 5.1. For the comparison, the average latency in the ledger data update transaction was used. Each latency was extracted from Figure 5.9 and 5.10.

Each gap between the left bar $L_{\text{Blockchain Core}}$ and the right bar L_{client} in each pair becomes the processing time in the client node including the network latency $L_{\text{gatewayNW}}$ between the client node and the blockchain. Since $L_{\text{gatewayNW}}$ is less than 1 millisecond in the simulation environment connected by ethernet, it can be negligible, but in practice if the network has very low bandwidth or if WAN is used, the latency in this interval should be examined carefully so as to avoid undesirable delay.

In those less than 150 clients, the margin between $L_{\text{Blockchain Core}}$ and L_{client} is trivial, but in the case of 150 and 200 clients, the gap mounts slightly by up to 0.043 seconds. In the case of 250 clients, the difference is evident. As the number of clients increases, so does the average latency in L_{client}. The inclination in L_{client} is bigger than in $L_{\text{Blockchain Core}}$ as the network sessions should be established in the client node. Because the latency, L_{client} in less clients are less affected by the delay in network connection, the gap between $L_{\text{Blockchain Core}}$ and L_{client} is small. On the contrary, the larger number of clients makes a big difference in latency caused by the network interface contention. However, the latency $L_{\text{Blockchain Core}}$ in blockchain is stable. With more than 250 clients, L_{client} soared drastically and with 350 clients the system went panic, but $L_{\text{Blockchain Core}}$ just showed a little increase in both cases.

5.4.2 Comparison with Other Study

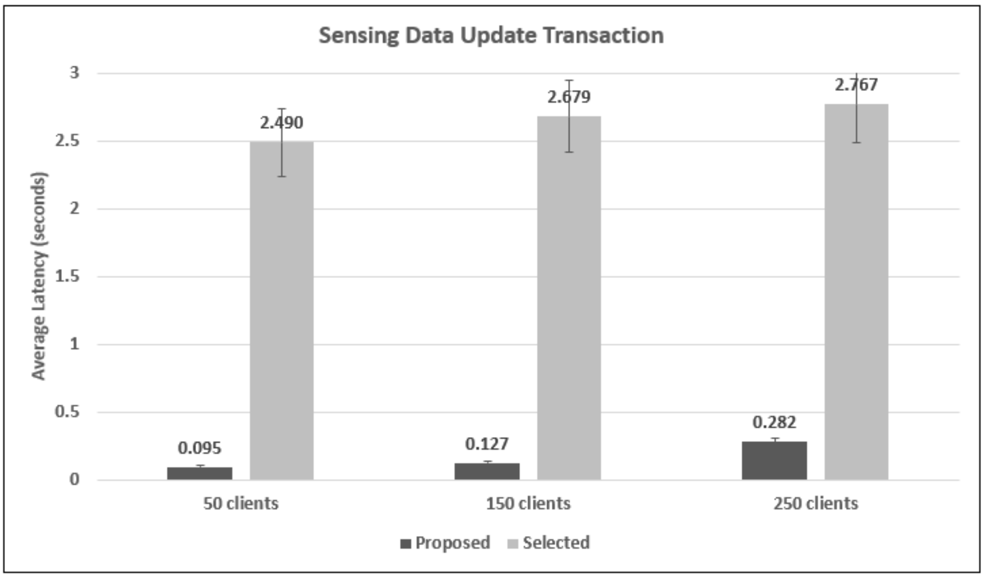

Figure 5.12: Average update transaction latency comparison

Among related works as shown in summarised Table 2.1 in Chapter 2, the work in [35] is closest to this research. It is based on the private blockchain. The differences are it is equipped with an IoT server for device management and a REST API server for interfaces.

For the comparison, the evaluation of the service execution time on storing sensing data in [35] is set as a benchmark, and the data update transaction latency in Figure 5.9 is used. Because query performance can be accelerated by indexing and has no latency caused by locks for updates, query transaction has less discrimination, so that update performance is compared.

In comparison with the work, the proposed architecture shows that it is faster up to 26 times in 50 clients and 9.8 times in 250 clients as shown in Figure 5.12.

The big difference of latency is mainly caused by the streamlined architecture. The proposed architecture excludes any additional management servers for IoT devices and middleware to relay IoT data. Instead, it only involves IoT gateways, on which client application SDK and connection API are implemented. Multiple IoT sensors are coupled with the dedicated IoT gateways by either wired or wireless connection, and IoT gateways transmit the sensed data directly to blockchain.

The interface with blockchain is based on gRPC. gRPC has better performance than any other data interface methods. Despite many advantages of gRPC, due to the limited TCP session, the simulation with over 300 clients was impossible. The analysis with regard to the interface methods is further discussed in Section 5.6.

In the proposed architecture, there is no centralised system component, so that no single point of failure exists. Therefore, the proposed architecture outperforms the existing study [35] in terms of latency as well as robustness.

5.5 System Resource Utilisation

In the experiments, Docker containers for each blockchain node on a single machine have been implemented as they represent virtually independent servers with many advantages such as ready to deployment without

Table 5.4: Hardware resource consumption per each node container (two peers)

COMPONENT	MEM (Max. MB)	MEM (Avg. MB)	CPU (Max. %)	CPU (Avg. %)
ORDERER	17.70	14.09	15.03	6.98
PEER-1	135.30	108.64	27.14	14.95
PEER-2	136.50	119.86	27.39	15.23
SUM	289.50	242.59	*69.56*	*37.16*

adjustment of computational environments. In practice, those containers can be distributed to multiple servers or each container can be placed on a dedicated server separately. Containers also can be replaced by the native installation without using virtualisation techniques. The deployment method and the resource assignment rely on business requirements and transaction volume in enterprise. Regarding the performed experiments, as mentioned in the comparative work [35], there are limitations in hardware size, which can be easily removed in practice. Nonetheless, as a benchmark model that can be extended, a base measurement of the resource consumption in the blockchain is still needed to size appropriate hardware for blockchain configurations.

Table 5.4, 5.5, and 5.6 show CPU and main memory consumption per a Docker container which hosts each blockchain component node. The tests with two, four, and six peer nodes with LevelDB were experimented, each of which used one ordering service and one CA host using state database update transaction. The host machine is equipped with i7-6700 CPU @ 3.40GHz and 8GB main memory.

When a chaincode is instantiated on a blockchain, it is launched by a new container for normally one endorsing peer node, and then if a transaction defined in the chaincode is submitted, new containers are created

Table 5.5: Hardware resource consumption per each node container (four peers)

COMPONENT	MEM (Max. MB)	MEM (Avg. MB)	CPU (Max. %)	CPU (Avg. %)
ORDERER	18.60	14.46	14.33	6.67
PEER-1	129.00	113.21	22.00	12.88
PEER-2	127.00	103.47	22.13	12.88
PEER-3	139.00	109.67	21.82	12.73
PEER-4	128.90	121.09	22.15	12.96
SUM	542.50	461.90	*102.43*	*58.11*

Table 5.6: Hardware resource consumption per each node container (six peers)

COMPONENT	MEM (Max. MB)	MEM (Avg. MB)	CPU (Max. %)	CPU (Avg. %)
ORDERER	19.60	15.58	15.41	7.24
PEER-1	148.30	111.82	19.60	11.85
PEER-2	145.40	109.56	20.32	11.99
PEER-3	145.60	111.67	19.05	11.82
PEER-4	148.10	115.94	18.74	10.63
PEER-5	143.50	122.92	19.09	10.79
PEER-6	139.30	122.06	19.31	10.43
SUM	889.80	709.55	*131.52*	*74.74*

dedicated to each endorsing peer node. These containers for chaincode instantiation consume very little system resources, and can be ignored in the light of main components. The CA node which controls membership and command line interaction node for administrators can also be neglected in terms of resource consumption as they are used only for a certain purpose occasionally. Therefore, only the usages of each peer node and ordering service node are investigated.

However, in practice, there are a few things more to be considered. If CouchDB is used, it runs on a discrete container, so that additional resources should be taken into account for the ledger DB implementation. The role of each peer node is also influential to the total resource consumption. Whether a peer is an endorser or not, or which endorsement policy is applied to the network affects the total performance, as it determines the utilisation of the peer node.

Each case of different peer numbers showed average 115 Mbytes memory consumption and around 15% CPU utilisation. The number of peers in the evaluation is just for the distinction of each container and the endorsement policy is set to 'OR', so that every container approximates in terms of the system utilisation.

Memory usage is related the application, which is a container in the experiment. When the application starts, it reserves memory area to some extent. Accordingly, the amount of memory usage is similar across all the cases.

In the case of CPU usages, however, the smaller number of peers shows a bit higher. It is associated with the endorsement policy. It is set to 'OR' among endorsers in the experiment based on the scenario, so that the transaction will be processed in one peer node while all the peer nodes interact with each other for the service discovery. Thus, all the submitted transactions are eventually spread out evenly to all peer nodes.

In contrast, in the case of 'AND' policy, since all the endorsing peer nodes should process the proposed transaction and verify it, the CPU usages in each peer node mount higher. Hence, in this case, it is evident that the more peers exist the worse the whole performance becomes as an inherent downside of decentralisation.

The orderer node relatively consumes less resources as it only intervenes the ordering of each transaction irrespective of the transaction validation or processing. In practice, the production systems adopt other consensus algorithms than 'Solo' that was used. The Solo implementation is mainly used for development purposes as it is not fault-tolerant. Other algorithms should configure multiple orderers and clusters, so that additional resources should be assigned.

In the case of six peers as shown in Table 5.6, each peer node consumes up to 150 Mbytes memory at its maximum usage, which sums up nearly 1 GBytes in total, whereas around 700 Mbytes in total is used on average. In the case of CPU utilisation, the total usage in italics is greater than 100% if the maximum usages of each peer node are accumulated. However, the utilisation expressed as a percentage cannot exceed 100%. Besides, since each peer node may discretely reach its own maximum CPU usage apart from others' behaviour, the accumulation might not be reasonable.

Nevertheless, during the peak time of a certain peer node, it may utilise CPU resource intensively, so that others are vulnerable to the shortage of the resources under multi-threading environments. This increases the CPU wait time, which in turn raises the likelihood of the contention for CPU. Therefore, it is still needed to consider each node's maximum usage for the predictive monitoring. Furthermore, if every node happens to utilise CPU at its maximum at the same time by any chance, the system must undergo serious delay in processing any transactions. In particular, the maximum CPU usage will be an important factor in hardware sizing. It should also be noted that the average CPU utilisation, more than 10%, is, in fact, quite high. Many systems in production are likely to maintain below 40% on average CPU usage in practice.

On the other hand, in the case of memory usage, the accumulated value is correct compared with CPU usage. It is simply because the resource utilisation methods are different. In terms of memory usage, applications allocate physical memory addresses when they start to run. In the experiment, the applications are the peer node containers. The applications are always up and running, so that regardless of user transactions, the reserved memory areas remain exclusively.

Thus, under the similar environments with such hardware resources as the experiment, it is recommended that fewer than six peers are to be implemented on a single host. Otherwise, an additional hardware resource should be deployed.

5.6 Further Analysis

Through the experiments, it has been found a system panic status that occurs when the number of simulated clients exceeded a certain limit, 300 clients. From the simulation with concurrent 250 clients upwards, one or two data streaming failure messages have been rarely returned and the occurrences became a bit more frequent with 300 clients. Then, with 350 clients, they soared. Subsequently, almost every transaction was timed out and could not run properly, which eventually caused the client node panic and any further processes were disabled.

The phenomenon was found both in data query and update simulation. Specifically, the streaming failure messages were very rarely returned both with 250 and 300 clients, while they were not found at all in less than 250 clients. Despite the messages, all the transactions were successfully processed in those simulations, but with 350 clients, the whole processes in the client node seemed to be hanged by the panic status. This phenomenon is mainly caused by the interface connection between the systems. The spikes in maximum latency in Figure 5.8 and 5.9 are closely related to the streaming failure. Therefore, the interface messaging protocol and causes of the panic state are analysed in this section.

5.6.1 Interface Messaging Protocols

General Classification

In distributed computing environments, each system should communicate with each other while they host diverse applications written in various kinds of programming languages and APIs, so that data integration is a huge workload to developers. There are a number of communication methods. Historically, Socket Programming [56], Remote Procedure Call (RPC) [57], Representational State Transfer (REST API) [8], and gRPC known as Google RPC [58] have been introduced in order and used widely up to the present.

Conventional RPC

RPC involves diverse applications such as Java Remote Method Invocation (RMI) [59], Common Object Request Broker Architecture (CORBA) [60], and Simple Object Access Protocol (SOAP) [61], which are a bit faded these days because they are not either collaborated by public communities or they are somewhat complicated to implement.

REST API

REST API defines web resources and uses HTTP/1.1 methods. It has been commonly used on the ground that the resources are defined intuitively and it can be implemented without any additional work due to the inheritance of HTTP. Despite the advantages, it has its limitations because it only declares the exchange format but has no standardised specifications, so that the parameters and response values are not explicit. Moreover, since it uses HTTP server modules, operational costs and risks on each computational node are enlarged as the modules should be deployed mostly by an additional server.

gRPC

However, gRPC uses protocol buffer, which is faster in data serialisation and more convenient to design API as the syntax is more readable compared with the descriptive JSON, so that it makes easier to develop APIs both in server and client side. In particular, it is based on HTTP/2 which enables many kinds of advantages than REST API based on HTTP/1.1. As a result, gRPC has lower system resource utilisation and higher performance than REST API. This is the reason why gRPC has been adopted to interface with each other in Hyperledger.

Although gRPC is used due to the above advantages, gRPC's transaction resubmission mechanism and TCP configuration in OS kernel bring about the aforementioned errors in the experiment.

5.6.2 Experiment Interface Limits

Linux kernel has several TCP configuration parameters. Among them, two parameters set the TCP connection limit on a single server.

- NET.IPV4.IP_LOCAL_PORT_RANGE (default 32768 – 61000 ports)
- NET.IPV4.TCP_FIN_TIMEOUT (default 60 seconds)

The former is for the maximum number of outbound sockets a host can create, and the latter is for the minimum time the sockets will stay.

By the above configuration with the default values, the number of TCP session that can be established in a second is 470 rounded-off by

$$(61000 - 32768) / 60 = 470.5$$

Thus, all the applications on a single server can establish up to less than 470 TCP connections concurrently in a second by OS default configuration.

The panic state occurred with 350 clients in the simulation. With 300 clients, 5.35% of the total query transactions and 5.39% of the total update transactions showed longer latency than 1 second, which were the cause of the spike in maximum latency. Hence, it is assumed that approximately 5% of the total transactions from 300 client simulation upwards are delayed, which means that those transactions in the 5% are to be resubmitted. The retry in gRPC is executed eight times concurrently by default until the session is connected.

Then, under the assumption of 95% throughput in a second, the total number of concurrently submitted transactions in 350 client simulation can be calculated as follows.

$$350 * 95\% = 332.5 \tag{5.3}$$
$$350 * 5\% * 8times = 140 \tag{5.4}$$
$$332.5 + 140 = 472.5 \tag{5.5}$$

The Equation 5.3 calculates the number of sessions that were successful in the first-try. The Equation 5.4 is for the number of sessions that were created by delayed and resubmitted transactions. The Equation 5.5 produces the total concurrent number of sessions. Therefore, in the 350 client simulation, up to 472 sessions may be opened, which is greater than 470 sessions by OS default. This caused the timeout and consequently the panic state.

For the case of 300 clients,

$$300 * 95\% = 285$$
$$300 * 5\% * 8times = 120$$
$$285 + 120 = 405$$

which is much less than the limit of 470.

5.6.3 Interface Method Implementation Comparison

In gRPC lifecycle, four different types of implementation methods exist. They are classified into unary RPC, Server streaming RPC, Client streaming RPC, and Bidirectional streaming RPC, based on the numbers of requests or responses and the message sending direction. The unary and the client streaming RPC are compared, as the client application sends a request to blockchain and then blockchain only responds to each request in the scenario.

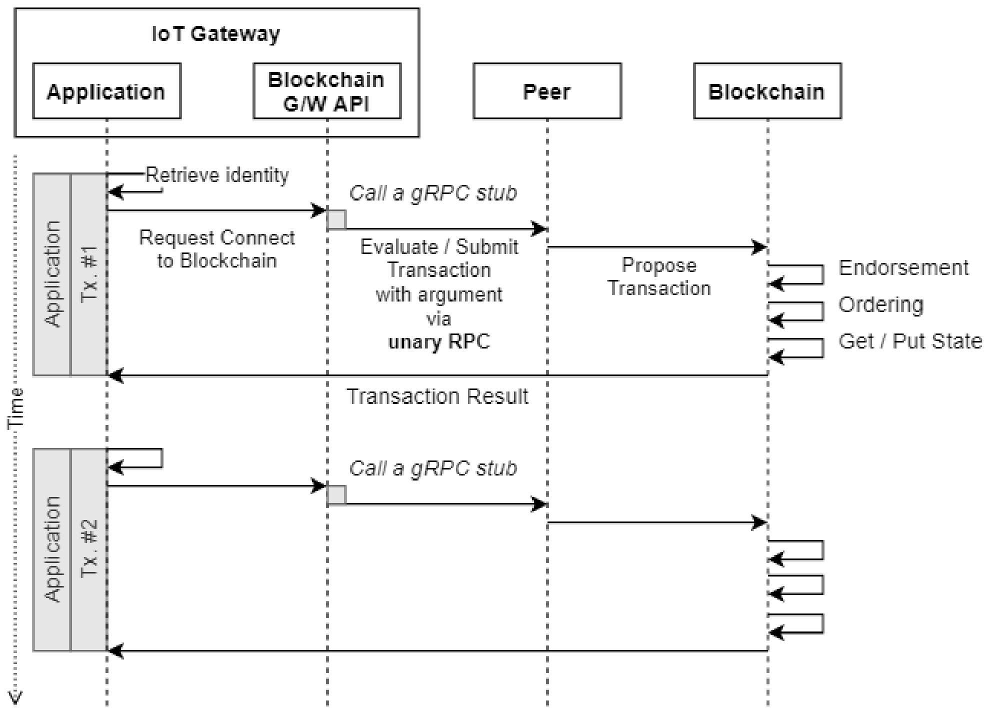

Figure 5.13: Transaction flow using unary RPC

Figure 5.13 depicts the usage of the experiment based on the unary RPC, which is a default implementation in the client SDK provided by Hyperledger.

In order to process IoT sensor data, the client application retrieves credentials from the digital wallet that holds public and private keys to access

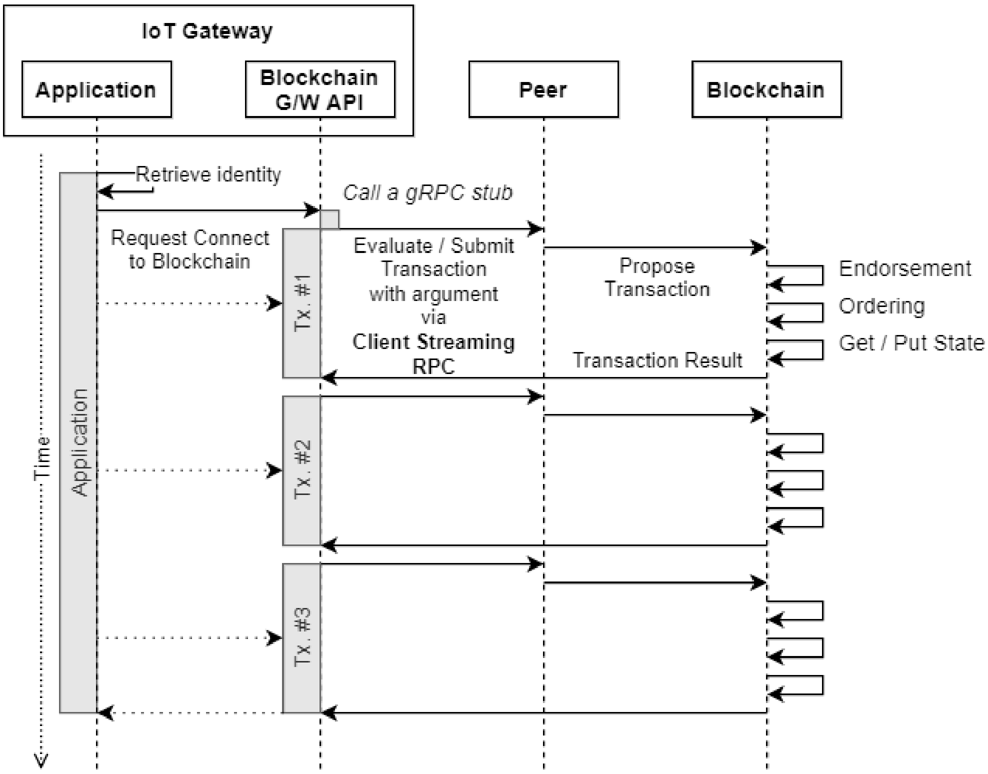

Figure 5.14: Transaction flow using client streaming RPC

blockchain in the first place. With the identity, the application requests the connection to blockchain via blockchain gateway API. The API then calls a gRPC stub to establish the connection. Once the connection is successful, the request flows as blockchain workflow. Based on the type of the request, blockchain can evaluate the query or submit the update on its channel via chaincode. In both cases, blockchain returns the result to the application. In this workflow, the unary RPC is used, and every transaction follows this lifecycle. Therefore, when each transaction is processed, a discrete connection between client and blockchain should be established. The more simulated clients are created, the more TCP sessions are needed, which is the cause of the problem discussed in the previous section.

In contrast, the client streaming RPC as shown in Figure 5.14 presents different way of session connection. It sends a stream of requests rather than a single request once the connection has been established. The server side, which is the blockchain in the architecture, does not necessarily respond after all the requests are transmitted. Since a client application uses an existing connection, there is no new connection set-up time incurred on the client side.

When the first transaction is invoked, the client application initially calls a gRPC stub with the identity, and then each transaction is processed onto the existing gRPC stub. The call for the stub does not occur at every time a transaction is submitted. The workflow in blockchain is the same as the unary RPC case. After all the transactions are processed, the client application completes. The limitation of TCP session in a single host does not apply. This implementation is efficient in terms of performance when the transaction is streaming input data to the server side regularly.

However, despite a number of strengths of gRPC, there exist weaknesses. The microservice implemented onto the gRPC stub tends to utilise the dedicated connection, so that load balancing is not guaranteed unless any appropriate configuration is set. In addition, gRPC itself does not typically support the browser, so that it cannot be called from the browser.

The unary RPC implementation was used as it was supposed to simulate multiple clients which in practice run on discrete machine. However, through the analysis of gRPC implementation methods, it is noted that client applications' interaction with blockchain can be far accelerated by using streaming RPC. Batch processing that iterates the same workflow with different input data corresponds to this streaming method.

5.7 Summary of the Chapter

Based on the proposed architecture, the model system has been tested in various ways in this chapter.

The test scenarios were highly refined by the structured methods both of functional and non-functional cases.

First, state database performance was compared between LevelDB and CouchDB. Second, the transaction processing performance for data query and update respectively was measured. Third, comparative evaluation in two ways, measurement by section and comparison with other work, was conducted. By the comparison, it has been proved that the proposed architecture outperforms other existing study.

Additionally, during the experiments, system resource utilisation was checked and critical problems in client interfaces were further analysed. These additional works will be considerably helpful especially when planning required hardware resources and designing the interfaces with external clients.

Chapter 6

Conclusion and Future Work

This chapter summarises and highlights the work performed in this thesis. Then, for further research and application of the proposed architecture, the associated future works are presented.

6.1 Conclusion of the Thesis

This thesis has shown how the blockchain platform can be utilised to integrate IoT data while high system performance is guaranteed. In order to demonstrate the concept, facility management system based on IoT sensor data in manufacturing was selected as a use case model. Both through functional and non-functional tests, the design was validated.

For the proposed architecture, a private blockchain platform, which can be applied to enterprise applications in IoT environments, has been developed. For the autonomous operation with the sensed data, the IoT gateway filters input data first and then the smart contract on blockchain processes the data as described in Section 3.3. The smart contract secures the IoT data integrity by hash functions and deals with delayed data by the timestamp check as described in Section 4.2. Finally, the input data update the ledger according to the selection criteria and invokes the actuator. Especially, both IoT gateway and private blockchain facilitated IoT data processing with more agility. Compliance issues with regard to data preservation was handled by using IPFS.

Interface methods for external entities and the limitations were introduced and analysed further in detail in Section 5.6. Blockchain is basically a network platform where all the participants share information, but for IoT environments, IoT devices were separated as clients from the network in the proposed architecture. This configuration reduces the burden of data volume, so that it makes the data process on blockchain faster. However, it needs authentication of each client and data interfaces. The authentication issue was solved by using digital credentials provided by a private blockchain. While other studies have deployed additional system components, such as a broker server or a REST API server, this research has employed direct streaming by gRPC through IoT gateways. This implies that the proposed architecture is more streamlined, so that it is more flexible and scalable. In addition, without any centralised components, it ensures the strengths of the decentralisation.

The approach written in this thesis surely enables the practitioners in the industry, who are going to develop blockchain to their businesses, to set up their initiatives and to make a progress while reducing the time of trial and error.

6.2 Future Work

In relation to the work done in this thesis, two different types of next steps are presented in this section.

One is a technological aspect, which is the extension of the applicable geographic location. The proposed architecture is simulated and examined in LAN environment. So as to facilitate the feature of the distributed computing, the system should run on WAN environment with a certain degree of performance guaranteed.

The other is a business application aspect, which is the extension of the proposed architecture to wider business area. Since the proposed design is the core part of blockchain network platform for IoT data processing, it can be applied in various environments using IoT data communication. Hardware parts SCM system is proposed as an immediate case of the application.

6.2.1 Verification in WAN

The proposed concept has been deployed in a LAN environment. If a multinational enterprise, for instance, intends to apply blockchain network to its offshore branches or geographically far away located business partners, the proposed architecture is quite effective in terms of decentralisation and simplicity in configuration. Nevertheless, the latency in endorsement and propagation of transaction requests should be taken into consideration. Therefore, further experiments are needed in WAN environments, where, if necessary, third-party solutions, such as network acceleration and other security appliances, may have to be collaborated to guarantee the compatibility and seamless transaction flow.

6.2.2 Extended Business Application

From business applications' perspective, the proposed architecture can be extended to machinery parts SCM system using blockchain for the whole machinery control and hardware parts life-cycle management.

Blockchain has been applied in many SCM use cases due to its features such as traceability and strengths in contract-based transactions [16][62]. These use cases are mostly reported in business-to-consumer (B2C) sales with an emphasis on P2P interactions. However, blockchain can demonstrate its strengths more in business-to-business (B2B) trades. In addition, the integration of applications among related different departments in an organisation demands blockchain platforms aside from B2B. Enterprise application interface (EAI) solutions have the role of such integration, but

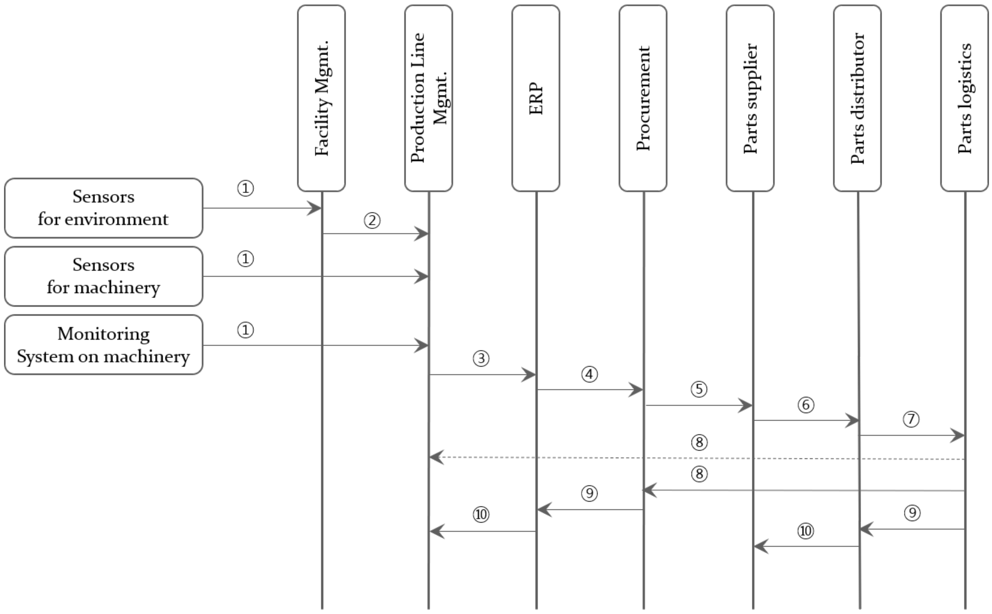

Figure 6.1: Data flow in hardware parts supply chain

they are based on centralisation with the same drawbacks.

In a B2B SCM, the processes such as purchasing, procurement, and vendor management are involved. These processes can be integrated on a private blockchain. Each stakeholder can communicate with each other while independent private communications among a certain group of participants are guaranteed by a channel. The privacy management on a private blockchain is efficient especially when a consortium should be set up.

Based on the proposed architecture, the health status of each hardware part in the machinery can be retrieved and utilised by capturing the health signal. Given data, smart contracts can determine the precautionary parts replacement, which is proactive maintenance and can reduce the unplanned service downtime. The procedures are integrated onto blockchain autonomously among related organisations or departments.

Figure 6.1 illustrates the data flow of the hardware parts supply chain management system. In brief, in this scenario, there are three stakeholders: owner of machinery, supplier of hardware parts, and logistics. The

Enterprise Resource Planning (ERP) system and the procurement system interact with other stakeholders for the purchase of hardware parts. All the stakeholders are integrated on the blockchain, but based on the characteristics of consortium, different network channels can be configured for data privacy.

The detailed data flow descriptions are as follows:

1. Sensing devices for circumstances and machinery transmit sensed data whenever they detect unusual symptoms based on pre-defined thresholds. Besides the IoT sensors, the machines can be equipped with their own embedded alert functions, so that they can send the system health data to production line management system.

2. Sensors monitoring environments can send the data through the facility management system to the production line management system. The first flow and the second flow are to be combined and processed by IoT gateways based on the proposed architecture. The production line management system can be simplified by IoT actuators and smart contracts on blockchain.

3. Every decision affecting the production is processed in the ERP system, and the parts are assets of a company, so that the properties are also stored and maintained in the ERP system. The transactions related to the changes in properties are processed on blockchain.

4. If a new part is to be deployed based on the decision in the ERP system, the purchasing task is processed in the procurement system, which interfaces with external partners on blockchain.

5. Purchased item list will be advertised to the parts suppliers via smart contracts on blockchain. When a notice of tender is placed, suppliers can build the consortium on a blockchain channel. When several projects need to be set up, the company can use a discreate channel for each project as well.

6. The parts supplier will check the requested items in their stock, if necessary, with the distributors. The parts supplier can configure its own blockchain with distributors and logistics. Otherwise, all of them can join the same blockchain with their sensitive interests reserved by private data functions on blockchain.

7. The supplier or the distributor orders the delivery to the logistics through a smart contract. Then, the requested parts are delivered to the production site.

8. The requested parts are placed to the production lines. Then, the delivery and installation information are recognised in the procurement system using a smart contract.

9. The procurement system updates the ERP system for a new part via a smart contract. Meanwhile, the completed delivery information will be sent to the parts distributor from the logistics via blockchain.

10. The parts distributor updates the supplier's database using the system that is used for the order. On the site, the ERP updates the status of the production line management system.

In this thesis, the autonomous control of machinery by using IoT devices on blockchain platform has been presented, and this workflow corresponds to the machine control stage. Likewise, the lifecycle of hardware parts involved in the production line can be self-maintained by using blockchain and smart contracts.

By applying blockchain and smart contracts, the hardware parts SCM system can be facilitated autonomously including parts ordering, delivery status tracking, purchase contracts, and payment. The provenance of the parts can be also traced. In addition, preservative maintenance is reinforced, which will minimise the unexpected service downtime.

Bibliography

[1] J. Mocnej, W. K. Seah, A. Pekar, and I. Zolotova, "Decentralised IoT architecture for efficient resources utilisation," *IFAC-PapersOnLine*, vol. 51, no. 6, pp. 168–173, 2018.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *white paper*, 2008.

[3] R. Kalis, "Using blockchain to validate audit trail data in private business applications," *University of Amsterdam*, June 2018.

[4] T. M. Fernández-Caramés and P. Fraga-Lamas, "A Review on the Use of Blockchain for the Internet of Things," *IEEE Access*, vol. 6, pp. 32979–33001, 2018.

[5] S. Guoqiang, C. Yanming, Z. Chao, and Z. Yanxu, "Design and Implementation of a Smart IoT Gateway," in *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pp. 720–723, 2013.

[6] D. Ryu, "Development of IoT Gateway based on Open Source H/W," *The Journal of the Korea institute of electronic communication sciences*, vol. 10, no. 9, pp. 1065–1070, 2015.

[7] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. K. Soong, and J. C. Zhang, "What Will 5G Be?," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 6, pp. 1065–1082, 2014.

[8] R. T. Fielding and R. N. Taylor, *Architectural styles and the design of network-based software architectures*, vol. 7. University of California, Irvine Irvine, 2000.

[9] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform," *white paper*, 2014.

[10] "Quorum." Available at `https://www.goquorum.com` (Accessed on 01/06/2020).

[11] "r3 | DLT & Blockchain Software Development Company." Available at `https://www.r3.com` (Accessed on 01/06/2020).

[12] "Hyperledger – Open Source Blockchain Technologies." Available at `https://www.hyperledger.org` (Accessed on 01/06/2020).

[13] A. M. Antonopoulos and G. Wood, *Mastering ethereum: building smart contracts and dapps*. O'reilly Media, 2018.

[14] 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)*, pp. 557–564, 2017.

[15] D. Puthal, N. Malik, S. P. Mohanty, E. Kougianos, and C. Yang, "The blockchain as a decentralized security framework [future directions]," *IEEE Consumer Electronics Magazine*, vol. 7, no. 2, pp. 18–21, 2018.

[16] A. Pal and K. Kant, "Using Blockchain for Provenance and Traceability in Internet of Things-Integrated Food Logistics," *Computer*, vol. 52, no. 12, pp. 94–98, 2019.

[17] S. Pongnumkul, C. Siripanpornchana, and S. Thajchayapong, "Performance Analysis of Private Blockchain Platforms in Varying Workloads," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–6, 2017.

[18] "Blockchain Charts." Available at `https://www.blockchain.com/charts/transactions-per-second` (Accessed on 01/06/2020).

[19] "Who Scales It Best? Inside Blockchains' Ongoing Transactions-Per-Second Race." Available at `https://bit.ly/2Xi95cz` (shortened url used) (Accessed on 01/06/2020).

[20] "Ethereum (ETH) Blockchain Explorer." Available at `https://etherscan.io` (Accessed on 01/06/2020).

[21] D. Mingxiao, M. Xiaofeng, Z. Zhe, W. Xiangwei, and C. Qijun, "A review on consensus algorithm of blockchain," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 2567–2572, IEEE, 2017.

[22] L. Bach, B. Mihaljevic, and M. Zagar, "Comparative analysis of blockchain consensus algorithms," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 1545–1550, IEEE, 2018.

[23] A. Dorri, S. S. Kanhere, and R. Jurdak, "Towards an optimized blockchain for IoT," in *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, pp. 173–178, ACM, 2017.

[24] S. Huh, S. Cho, and S. Kim, "Managing IoT devices using blockchain platform," in *2017 19th International Conference on Advanced Communication Technology (ICACT)*, pp. 464–467, Feb 2017.

[25] R. B. Chakraborty, M. Pandey, and S. S. Rautaray, "Managing Computation Load on a Blockchain-based Multi-Layered Internet-of-Things Network," *Procedia computer science*, vol. 132, pp. 469–476, 2018.

[26] S. Cho and S. Lee, "Survey on the Application of BlockChain to IoT," in *2019 International Conference on Electronics, Information, and Communication (ICEIC)*, pp. 1–2, IEEE, 2019.

[27] A. Boudguiga, N. Bouzerna, L. Granboulan, A. Olivereau, F. Quesnel, A. Roger, and R. Sirdey, "Towards better availability and accountability for IoT updates by means of a blockchain," in *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 50–58, IEEE, 2017.

[28] S. Tayeb, S. Latifi, and Y. Kim, "A survey on IoT communication and computation frameworks: An industrial perspective," in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 1–6, IEEE, 2017.

[29] E. Ko, J. Kang, and J. Park, "A middleware for smart object in ubiquitous computing environment," in *2012 8th International Conference on Computing Technology and Information Management (NCM and ICNIT)*, vol. 1, pp. 400–403, IEEE, 2012.

[30] M. Yannuzzi, R. Milito, R. Serral-Gracià, D. Montero, and M. Nemirovsky, "Key ingredients in an IoT recipe: Fog Computing, Cloud computing, and more Fog Computing," in *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pp. 325–329, Dec 2014.

[31] A. Dusia, Y. Yang, and M. Taufer, "Network quality of service in docker containers," in *2015 IEEE International Conference on Cluster Computing*, pp. 527–528, IEEE, 2015.

[32] L. Wang and R. Ranjan, "Processing Distributed Internet of Things Data in Clouds," *IEEE Cloud Computing*, vol. 2, no. 1, pp. 76–80, 2015.

[33] B. Liu, X. L. Yu, S. Chen, X. Xu, and L. Zhu, "Blockchain Based Data Integrity Service Framework for IoT Data," in *2017 IEEE International Conference on Web Services (ICWS)*, pp. 468–475, June 2017.

[34] X. Liang, J. Zhao, S. Shetty, and D. Li, "Towards data assurance and resilience in IoT using blockchain," in *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, pp. 261–266, Oct 2017.

[35] L. Hang and D.-H. Kim, "Design and implementation of an integrated IoT blockchain platform for sensing data integrity," *Sensors*, vol. 19, no. 10, p. 2228, 2019.

[36] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, and et al., "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[37] J. Wan, J. Li, M. Imran, and D. Li, "A blockchain-based solution for enhancing security and privacy in smart factory," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3652–3660, 2019.

[38] "IPFS – Powers the Distributed Web." Available at `https://ipfs.io` (Accessed on 01/06/2020).

[39] J. Benet, "IPFS – content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.

[40] "Git." Available at `https://git-scm.com` (Accessed on 01/06/2020).

[41] Y. Chen, H. Li, K. Li, and J. Zhang, "An improved p2p file system scheme based on ipfs and blockchain," in *2017 IEEE International Conference on Big Data (Big Data)*, pp. 2652–2657, IEEE, 2017.

[42] A. Haroon, M. A. Shah, Y. Asim, W. Naeem, M. Kamran, and Q. Javaid, "Constraints in the IoT: the world in 2020 and beyond," *(IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 7, no. 11, pp. 252–271, 2016.

[43] S. Al-Sarawi, M. Anbar, K. Alieyan, and M. Alzubaidi, "Internet of Things (IoT) communication protocols: Review," in *2017 8th International Conference on Information Technology (ICIT)*, pp. 685–690, May 2017.

[44] "Docker." Available at `https://www.docker.com` (Accessed on 01/06/2020).

[45] "Hyperledger Fabric SDK for node.js Module." Available at `https://hyperledger.github.io/fabric-sdk-node/release-1.4/module-fabric-network.html` (Accessed on 01/06/2020).

[46] "Hyperledger Caliper." Available at `https://hyperledger.github.io/caliper` (Accessed on 01/06/2020).

[47] "Hyperledger Explorer." Available at `https://www.hyperledger.org/projects/explorer` (Accessed on 01/06/2020).

[48] "Apache JMeter - Apache JMeter™." Available at `https://jmeter.apache.org` (Accessed on 01/06/2020).

[49] M. C. Loring, M. Marron, and D. Leijen, "Semantics of Asynchronous JavaScript," in *Proceedings of the 13th ACM SIGPLAN International*

Symposium on on Dynamic Languages, DLS 2017, (New York, NY, USA), p. 51–62, Association for Computing Machinery, 2017.

[50] S. Muralidharan and H. Ko, "An InterPlanetary File System (IPFS) based IoT framework," in *2019 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 1–2, Jan 2019.

[51] "Functional Testing: A Complete Guide with Types and Example." Available at `https://www.softwaretestinghelp.com/guide-to-functional-testing` (Accessed on 01/06/2020).

[52] "A Complete Non-Functional Testing Guide for Beginners." Available at `https://www.softwaretestinghelp.com/what-is-non-functional-testing` (Accessed on 01/06/2020).

[53] "Hyperledger-fabricdocs master documentation." Available at `https://hyperledger-fabric.readthedocs.io/en/release-1.4` (Accessed on 01/06/2020).

[54] "Apache CouchDB." Available at `https://couchdb.apache.org` (Accessed on 01/06/2020).

[55] "Google LevelDB." Available at `https://github.com/google/leveldb` (Accessed on 01/06/2020).

[56] W. R. Stevens and T. Narten, "Unix network programming," *ACM SIGCOMM Computer Communication Review*, vol. 20, no. 2, pp. 8–9, 1990.

[57] A. D. Birrell and B. J. Nelson, "Implementing remote procedure calls," *ACM Transactions on Computer Systems (TOCS)*, vol. 2, no. 1, pp. 39–59, 1984.

[58] "gRPC – A high-performance, open source universal RPC framework." Available at `https://grpc.io` (Accessed on 01/06/2020).

[59] "RMI – Remote Method Invocation." Available at https://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html (Accessed on 01/06/2020).

[60] "CORBA – Welcome To CORBA Web Site." Available at https://www.corba.org (Accessed on 01/06/2020).

[61] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple object access protocol (SOAP) 1.1," 2000.

[62] F. Tian, "An agri-food supply chain traceability system for China based on RFID & blockchain technology," in *2016 13th international conference on service systems and service management (ICSSSM)*, pp. 1–6, IEEE, 2016.

Appendices

Appendix A

Usage of PoC Library

A.1 Git Repository for PoC Software Library

Clone the Git repository
`https://gitlab.ecs.vuw.ac.nz/yup/mepoc1.git`

A.2 Prerequisites for the Deployment

1. Ubuntu 18.04.3 LTS
2. Docker Docker version 17.06.2-ce or greater
3. Node.js version 10.x is recommended
4. The Fabric Node.js SDK requires Python 2.7 instead of Python 3.5
5. Go Programming Language version 1.12.x or greater
6. Export GOPATH

   `export GOPATH=$PoC_HOME/go`
7. Hyperledger Fabric platform-specific binaries for the specified Fabric version under $PoC_HOME/bin (included in the Git repository)

`configtxgen, configtxlator, cryptogen, discover, idemixgen, orderer, peer, fabric-ca-client`

A.3 Deployment Procedure

1. Generate certificates and keys

   `$PoC_HOME/basic-network/generate.sh`

2. Modify certificate file path in

   `$PoC_HOME/basic-network/connection.json`

3. Modify certificate file path in

   `$PoC_HOME/basic-network/docker-compose.yml`

A.4 Start & Stop Blockchain

1. Start

   `$PoC_HOME/basic-network/start-p4-level.sh`

2. Stop

   `$PoC_HOME/basic-network/teardown-p4.sh`

Appendix B

Test Scenarios and Report

B.1 Functional Test

1. **Unit Test** : check for required parameters and return value

   1.1. Evaluate / submit each function with incorrect number of arguments

   1.2. Input string type of integer type as an argument

   1.3. Input an argument with long characters (up to 32-bit words)

   1.4. Read the message from evaluate / submit function

   1.5. Read the return values from evaluate / submit function

2. **Smoke Test** : build verification

   2.1. Execute blockchain start script

   2.2. Execute cryptographic certificate file generation script

   2.3. Find the public and private keys in the file system

   2.4. Execute blockchain teardown script

   2.5. Find the source codes of chaincode in the mounted volume

   2.6. Execute ”go build” command

   2.7. Find the executables in the file system
   2.8. Install the chaincode

3. **Sanity Test** : major and vital functionalities working
   3.1. Initialise a chaincode on a channel
   3.2. Execute query function on blockchain
   3.3. Execute query application using client SDK to access blockchain
   3.4. Execute listening application using client SDK to receive Event messages

4. **Regression Test** : code change having not adversely affected existing features
   4.1. Insert a simple text message in the chaincode and read the console
   4.2. Upgrade the above 4-1 chaincode and then execute it
   4.3. With the instantiated 4-2 chaincode, iterate the unit test

5. **Integration Test** : multiple functional modules working together
   5.1. Using generated key, access blockchain (execute query function) in client
   5.2. Execute query function to check the current actuator status
   5.3. Execute update function as a client and check the console of listening client

6. **Usability Test** : for production usage by users (similar to UAT)
   6.1. Input modified value for at least one argument using update function
   6.2. Input modified hash value using update function

6.3. Input string type argument for signal data using update function

6.4. Input unregistered sensor ID using update function

6.5. Input modified value for time argument using update function

6.6. Input normal data using update function

6.7. Input modified value (bigger than time threshold) for time argument using update

6.8. Execute query function on blockchain with a specific sensor ID

6.9. When actuator disabled, input signal data over threshold, and check status change

6.10. When actuator enabled, input signal data over threshold, and check status change

6.11. When actuator enabled, input signal data below threshold, and check status change

6.12. When actuator disabled, input signal data below threshold, and check status change

B.2 Non-Functional Test

1. **Performance Test** : evaluate the overall performance of the system

   1.1. Run Caliper benchmark performance test tool (250 devices) – Query / Update

   1.2. Run Caliper benchmark performance test tool (250 devices) – Query / Update 50 times

2. **Stress / Load Test** : validate that the system performs as expected under stress

2.1. Run Caliper benchmark performance test tool (50 devices) – Device Register
2.2. Run Caliper benchmark performance test tool (100 devices) – Device Register
2.3. Run Caliper benchmark performance test tool (150 devices) – Device Register
2.4. Run Caliper benchmark performance test tool (200 devices) – Device Register
2.5. Run Caliper benchmark performance test tool (250 devices) – Device Register
2.6. Run Caliper benchmark performance test tool (50 devices) – Data update
2.7. Run Caliper benchmark performance test tool (100 devices) – Data update
2.8. Run Caliper benchmark performance test tool (150 devices) – Data update
2.9. Run Caliper benchmark performance test tool (200 devices) – Data update
2.10. Run Caliper benchmark performance test tool (250 devices) – Data update

3. **Volume Test** : verify when a large volume of data are involved

3.1. Run Caliper benchmark performance test tool (100 records) – Data Query
3.2. Run Caliper benchmark performance test tool (300 records) – Data Query
3.3. Run Caliper benchmark performance test tool (500 records) – Data Query

4. **Compatibility Test** : evaluate that the application is compatible with others
    4.1. Upgrade chaincode to different version and check the unit test
    4.2. Check if the channel capability is set
5. **Recovery Test** : verify against Hardware and Software failures
    5.1. Check the existence of necessary initial files
    5.2. Delete all the configuration and source files except initial files, and restore the system
6. **Failover Test** : verify in case of a system failure
    6.1. Stop running one peer container and check query function
    6.2. Stop running two peer containers and check query function
7. **Security Test** : ensure that the application has no loopholes
    7.1. Execute client application by ID with public and private keys
    7.2. Execute client application by ID without keys
    7.3. Execute client application by ID with manipulated keys
8. **Scalability Test** : verify the application is capable of increased requests
    8.1. Add a peer node and check it is running on blockchain
    8.2. Add an orderer node and check it is running on blockchain
    8.3. Add an organisation and check it is interaction on blockchain
    8.4. Add a channel and attach peers, then check it works
9. **Localisation Test** : verify the application in different languages
    9.1. Insert data that use 2-byte form character set to blockchain

9.2. Query data that use 2-byte form character set to blockchain

10. **benchmark Test** : use a base for any new application

10.1. Set up a baseline for performance latency compared with other use cases

B.3 Test Report Template

Hyperledger Caliper Test Report Sample (Figure B.1)

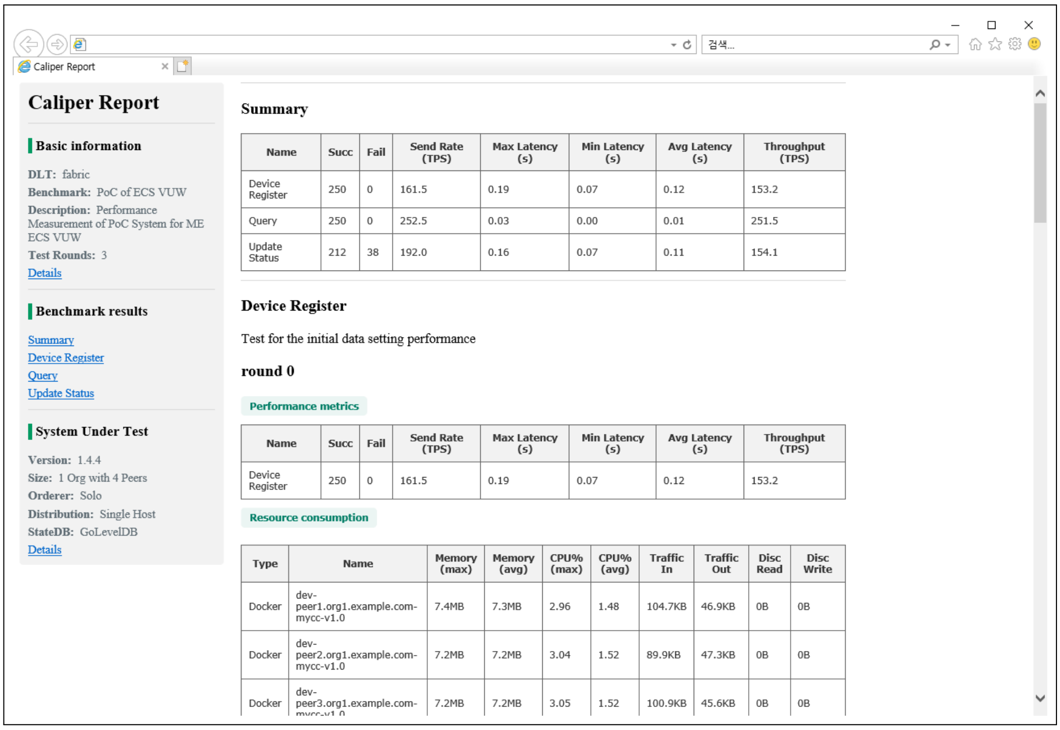

Figure B.1: Hyperledger Caliper Report