Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

Dissertação

Using Blockchain for IoT Decentralization

Diogo Pais Solipa

Orientador(es) | Pedro Salgueiro
José Saias

Évora 2025

Using Blockchain for IoT Decentralization

Diogo Pais Solipa

Orientador(es) | Pedro Salgueiro
José Saias

Évora 2025

Acknowledgements

This work represents the convergence of several efforts, and as such, I would like to begin by expressing my deepest gratitude to my parents for providing me with the opportunity to pursue higher education, and for their constant insistence and encouragement, which allowed me to complete this chapter of my life.

To Cristiana, for always being an active voice reminding me that when we start something, we see it through to the end, and for being present at every stage of this journey.

To my uncles, Mário and Cláudia, who, on every family occasion, were relentless in their motivation to make things happen, to the point of winning me over through sheer persistence.

Finally, to Professor Pedro and Professor José, without whom this work would undoubtedly not exist, for their willingness to continue supervising this dissertation after an almost two-year pause, and for being the pillars that sustained this effort.

Thank you all for making me believe in myself and for giving me motivation when it was scarce. I sincerely hope to be able to return this support in the future.

Contents

List of Figures

List of Tables

List of Acronyms

AI	*Artificial Intelligence*
API	*Application Programming Interface*
BaaS	*Blockchain-as-a-Service*
BFT	*Byzantine Fault Tolerance*
CA	*Certificate Authority*
CLI	*Command-Line Interface*
CPU	*Central Processing Unit*
DAG	*Directed Acyclic Graph*
DBMS	*Database Management System*
DBFT	*Delegated Byzantine Fault Tolerance*
DLT	*Distributed Ledger Technology*
ETL	*Extract, Transform, Load*
FBA	*Federated Byzantine Agreement*
GPU	*Graphics Processing Unit*
gRPC	*Google Remote Procedure Call*
HLF	*Hyperledger Fabric*
HTTP	*HyperText Transfer Protocol*
ICT	*Information and Communication Technology*
IoMT	*Internet of Medical Things*
IoT	*Internet of Things*
JSON	*JavaScript Object Notation*
KPI	*Key Performance Indicator*
LTS	*Long-Term Support*

MSP	*Membership Service Provider*
PBFT	*Practical Byzantine Fault Tolerance*
PKI	*Public Key Infrastructure*
PoA	*Proof of Activity*
PoET	*Proof of Elapsed Time*
PoI	*Proof of Importance*
PoL	*Proof of Location*
PoS	*Proof of Stake*
PoW	*Proof of Work*
RAFT	*Reliable, Replicated, Redundant and Fault Tolerant*
RAM	*Random Access Memory*
REST	*Representational State Transfer*
SDK	*Software Development Kit*
SOA	*Service-Oriented Architecture*
TPS	*Transactions Per Second*

Abstract

The integration of Internet of Things (IoT) and Blockchain technologies presents a compelling opportunity to address longstanding challenges related to data integrity, trust, and decentralization in distributed systems. Blockchain is a specific implementation of distributed ledger technologies (DLTs), known for its security, trust, and decentralization. IoT, on the other hand, has become increasingly relevant, generating vast amounts of data and requiring high uptime for connected devices, which are susceptible to data tampering, privacy breaches, and single points of failure. Blockchain, and more broadly Distributed Ledger Technologies (DLTs), offer a decentralized approach to securing data exchanges and ensuring system resilience.

This dissertation explores the integration of Blockchain with an IoT network, whilst also exploring other DLT specifications such as Tangle, Hashgraph, Sidechains, and Holochain, assessing their suitability for integration within IoT architectures. Particular emphasis is placed on Hyperledger Fabric, a permissioned blockchain framework tailored for enterprise-grade applications. A complete permissioned network was developed, including customized smart contracts, chaincode, and a REST gateway to simulate the exposition to real world IoT interactions. Two state database configurations, LevelDB and CouchDB, were analysed to evaluate trade-offs between performance and query flexibility. Standard benchmarking was performed using Hyperledger Caliper, followed by IoT tailored experiments employing real sensor data.

The results show that Hyperledger Fabric can sustain stable throughput, low latency, and full transactional reliability across different workloads. LevelDB exhibited higher performance in ingestion intensive scenarios, whereas CouchDB provided greater analytical capability through JSON-based rich queries. The adopted data model, based on composite keys (sensorID:timestamp), proved scalable and efficient for time-ordered sensor readings.

Overall, the findings validate the viability of Hyperledger Fabric as a backbone for decentralized IoT data infrastructures, capable of ensuring integrity and scalability while supporting modular extensions for analytics and real time applications.

Keywords: IoT, Blockchain, Distributed Systems, State Management, Decentralization.

Resumo

Utilização da Blockchain para a Descentralização da IoT

A integração das tecnologias IoT e Blockchain representa uma oportunidade para abordar desafios persistentes relacionados com a integridade dos dados, a confiança e a descentralização em sistemas distribuídos. A Blockchain é uma implementação específica das Distributed Ledger Technologies (DLTs), reconhecida pelas suas propriedades de segurança, confiança e descentralização. A IoT tem vindo a crescer, gerando grandes volumes de dados e exigindo elevada disponibilidade de dispositivos, que são vulneráveis a adulteração de dados, quebras de privacidade e pontos únicos de falha. A Blockchain, e mais amplamente as DLTs, oferecem uma abordagem descentralizada para garantir a segurança das trocas de dados e a resiliência do sistema.

Esta dissertação explora a integração da Blockchain com uma rede IoT, analisando outras abordagens como Tangle, Hashgraph, Sidechains e Holochain, avaliando a sua adequação à integração em arquiteturas IoT. É dada especial ênfase à Hyperledger Fabric, uma framework de blockchain privada, direcionada para aplicações empresariais. Foi desenvolvida uma rede privada, os respectivos smart contracts, e uma REST Gateway, para simular a exposição a interações IoT do mundo real. Foram analisadas duas configurações da base de dados de estado, LevelDB e CouchDB, de modo a avaliar os compromissos entre desempenho e flexibilidade de consulta. O benchmarking foi realizado com o Hyperledger Caliper.

Os resultados demonstram que a Hyperledger Fabric consegue manter um throughput estável, baixa latência e total fiabilidade transacional sob diferentes cargas de trabalho. A LevelDB apresentou melhor desempenho em cenários intensivos de inserção de dados, enquanto a CouchDB ofereceu maior capacidade analítica através de consultas ricas baseadas em JSON. O modelo de dados adotado, baseado em chaves compostas (sensorID:timestamp), revelou-se escalável e eficiente para leituras de sensores ordenadas temporalmente.

Em suma, os resultados obtidos validam a viabilidade da Hyperledger Fabric como infraestrutura base para sistemas descentralizados de dados IoT, capazes de assegurar integridade e escalabilidade, enquanto suportam extensões modulares para análise e aplicações em tempo real.

Palavras-chave: IoT, Blockchain, Distributed Systems, State Management, Decentralization.

1

Introduction

The applications which integrate Blockchain and IoT have grown significantly, with an increase relevance in this research area, due to the complementarity between the two technologies. This dissertation aims to explore how these systems can be interconnected, portraying some of the existing implementations, their architectures, and particularities, such as security, decentralization, and trust.

1.1 Context

Blockchain technology gained widespread attention with the advent of Bitcoin [Nak08], which became the most prominent cryptocurrency over the past decade. Its core principles, security, decentralization, and trust, are achieved through consensus and cryptographic algorithms that ensure the immutability and integrity of transactions over the network. Although cryptocurrencies were the initial use case, blockchain applications have expanded to a wide range of domains, including finance, healthcare, energy, and public services [KAMA22].

Blockchain can be considered a specific implementation of Distributed Ledger Technology (DLT), where data records are decentralized across a network of nodes. DLTs ensure that all participants in the network hold a copy of the ledger, therefore making unauthorized alterations increasingly difficult, unlike traditional centralized databases. The Section 1.1 dedicated to DLTs further depicts on the different approaches and architectural variants.

In parallel, IoT has become increasingly common in daily life, with applications ranging from smart homes to industrial monitoring. These systems typically involve networks of numerous devices that continuously generate and exchange data. Such systems pose challenges in terms of data authenticity, high availability, resource usage, and real-time processing [PVM24].

One of the central issues in IoT is ensuring the integrity and reliability of the data being exchanged. DLTs present a compelling alternative to the existing centralized solutions, as their decentralized nature enables trust among distributed actors without a single point of failure. Every node stores a replica of the data, and consensus mechanisms validate transactions before they are committed, ensuring tamper-resistance.

This work provides a comparative overview of different Blockchain implementations and examines their capacity to support IoT architectures. The focus lies in understanding how these technologies can be combined to meet the constraints of IoT environments while leveraging blockchain's trust mechanisms.

1.2 Motivation

Blockchain applications exist in various industries, however, in 2025, the biggest and most used applications exist in the financial services industry, in the form of cryptocurrencies. Its development was a big leap into building systems based on trustworthy transactions, possible alternatives to centralized currencies such as the euro or the dollar, whilst being supported by cryptographic algorithms and without a need of a third party element to oversee, aid and conduct the process. Although the other industries applications are possibly not as disruptive as cryptocurrencies, there are plenty of use cases that could make use of a tamper resistant database, fully distributed and accessed by any client. IoT shares most of these features and concerns. Most systems are distributed geographically with sensors present in various parts of the globe, streaming data into databases, which presents an issue of data validity. Blockchain is a contender to solve this issue.

Furthermore, in the IoT context there are several concerns regarding battery life, with typically low resources available for mobile devices. Blockchain however, due to all its security algorithms, isn't the most performant of databases. This presents a opportunity to address how can Blockchain be used for high-load scenarios, and how will different architectural changes impact its performance, and if it could indeed work as a real alternative to centralized solutions focused on data storage and access.

1.3 Objectives

This work aims to explore the integration of Blockchain in IoT ecosystems, particularly focusing on environments characterized by continuous, high volume data streams. While Blockchain offers decentralization, immutability, and secure transaction recording, it also raises concerns regarding performance, latency, and scalability under demanding workloads.

The goal is to identify and evaluate solutions that enable Blockchain to operate effectively within these constraints. This includes testing different architectural setups, state databases, and access strategies using benchmarking tools such as Hyperledger Caliper. By analysing the behaviour of the network under various conditions, we seek to understand the trade-off between performance, flexibility, and resilience.

Additionally, this work explores the feasibility of real time, on-chain data visualization, an area often underdeveloped in current Blockchain systems. We aim to assess not only its technical implementation but also its impact on network usability and system load, particularly in scenarios where responsive feedback is critical.

Ultimately, this dissertation strives to contribute a clearer perspective on how Blockchain can be tailored to support secure, scalable, and transparent data flows in IoT contexts.

1.4 Document Structure

This dissertation is organized into five Chapters, each contributing to a progressive understanding of the integration between Blockchain and the Internet of Things (IoT), from conceptual foundations to system implementation and experimental validation.

Chapter 2 presents the theoretical and conceptual background. It introduces Distributed Ledger Technologies (DLTs), with emphasis on Blockchain and its core components such as consensus mechanisms, network types, and architectures. The Chapter also explores the Hyperledger ecosystem, particularly Fabric, and discusses its relevance to enterprise-grade applications. It concludes with an overview of IoT architectures, integration challenges, and existing research efforts combining IoT with Blockchain.

Chapter 3 details the proposed system, defining the architectural rationale, design requirements, and implementation process. It begins with the specification of objectives and requirements, followed by the data and interaction model and the rationale behind key architectural decisions. The Chapter then describes the system's composition, including the Hyperledger Fabric network configuration, smart contract development, and client-gateway integration, demonstrating how the proposed blockchain infrastructure supports IoT data ingestion, traceability, and validation.

Chapter 4 presents the experimental evaluation of the system. It describes the datasets, metrics, and benchmarking methodologies adopted to assess network performance and scalability. Standard benchmarking using Hyperledger Caliper is performed under different state database configurations, LevelDB and CouchDB, followed by IoT tailored evaluations employing real sensor data and custom chaincode. Comparative analyses of performance, latency, throughput, and query behaviour are discussed, highlighting trade-off between performance efficiency and analytical flexibility.

Chapter 5 concludes the dissertation by summarizing the key findings and relating them to the initial research objectives. Outlines limitations encountered during implementation, and proposes directions for future research, including large-scale deployments, alternative consensus mechanisms, and the incorporation of real time data streams and hybrid architectures.

2

Concepts and State of the Art

This Chapter maps a broad image of the landscape of Distributed Ledger Technologies (DLTs), more specifically, on the blockchain implementation of the ledger. It also provides a brief overview of the growth, applications, and general architecture of the IoT. Furthermore, we develop the idea that the integration of blockchain with IoT networks can solve many of the existing challenges associated with centralized platforms and data authenticity. Lastly, we provide literature based examples of related work and their respective results, as well as other relevant insights, such as metrics or data used.

2.1 Theoretical Background

In section 2.1 we have presented a roadmap to better understand the evolution of DLTs, their different implementations, highlighting blockchain and its integration with IoT over their lifespan. We described various consensus mechanisms, their impacts and differences when integrated in the different types of blockchain, Permissioned and Permissionless. Specifically, we focused on the Hyperledger ecosystem, detailing Fabric and Indy, two permissioned networks.

2.1.1 Distributed Ledger Technologies

Since it has been known, humans have been using ledgers to keep track of valuable and important information like money transactions, property owners, or any other sensitive data. With the development of technology, these paper and book ledgers have now become digital, in a way similar to what we know today as databases. A Distributed Ledger is not just a digital ledger, but one that is distributed out through an online network of participants, that can be anywhere and accessed at any time [SP22]. Within this network all participants have a copy of the ledger and any changes on that same ledger must be mirrored across all existing copies, allied with cryptographic functions like the hash function as a hand print for ledger elements, any change must be done by mutual agreement of the participants, which is called the consensus algorithm. Blockchain is a specification of a Distributed Ledger with some particularities that make it as hyped and used as of today, described in this section.

Apart from blockchain there are many types of implementations of distributed ledgers, each implementing specific consensus algorithms, or attempting to fix the biggest concerns around blockchain, be it added security, scalability, or performance. In the following sections we will provide various examples of these implementations.

Tangle

One of the biggest issues with Blockchain is the need for high power systems to operate efficiently [SZJS22]. On the IoT landscape it's known that the available power on mobile devices is relatively low compared to big cloud servers. In this regard, Tangle comes as an alternative for a Blockchain implementation. While a Blockchain could be represented as a linked list data structure, Tangle is based on a Direct Acyclic Graph (DAG). Each DAG node will represent a transaction, as a block in Blockchain, and the connections between nodes (transactions), would represent the validation of transactions [EIP18]

Hashgraph

Hashgraph is a consensus algorithm for asynchronous Byzantine fault tolerance with its focus on distributed ledgers. What makes it a stand out from other similar algorithms is its ability to achieve consensus without needing to exchange any more messages, since participant's votes can be calculated with public information from the network, there's no need to re-broadcast votes [Cra21].

Sidechain

A Sidechain is in its core another Blockchain. It was developed as a new layer that could improve performance, security, or any other specification desired. By adding a secondary chain, sidechain, to the original Blockchain, completely different consensus mechanisms can be implemented from those being used by the original chain, adding new functionalities, and adding another layer of security. If a sidechain is corrupted, since they are isolated and self-contained, any damage is only done to the sidechain and never to the primary chain [SCP$^+$20].

HoloChain

One of the biggest issues regarding vanilla Blockchain, is its scalability issues. In a standard Blockchain, every node needs to store and transmit valid data, making it in some scenarios, inefficient and slow. Holochain is a peer-to-peer networking protocol to develop serverless applications [KPBC22]. Each participant, or user, will run a Holochain on its own device, storing predominantly only the users data, having no need to store and validate all the networks data. With a set of immutable rules, its able to store minimal data from the other networks, making it extremely efficient, and viable for mobile devices, which is an very interesting perspective for IoT.

Figure 2.1 aggregates some of the existing types of DLTs.

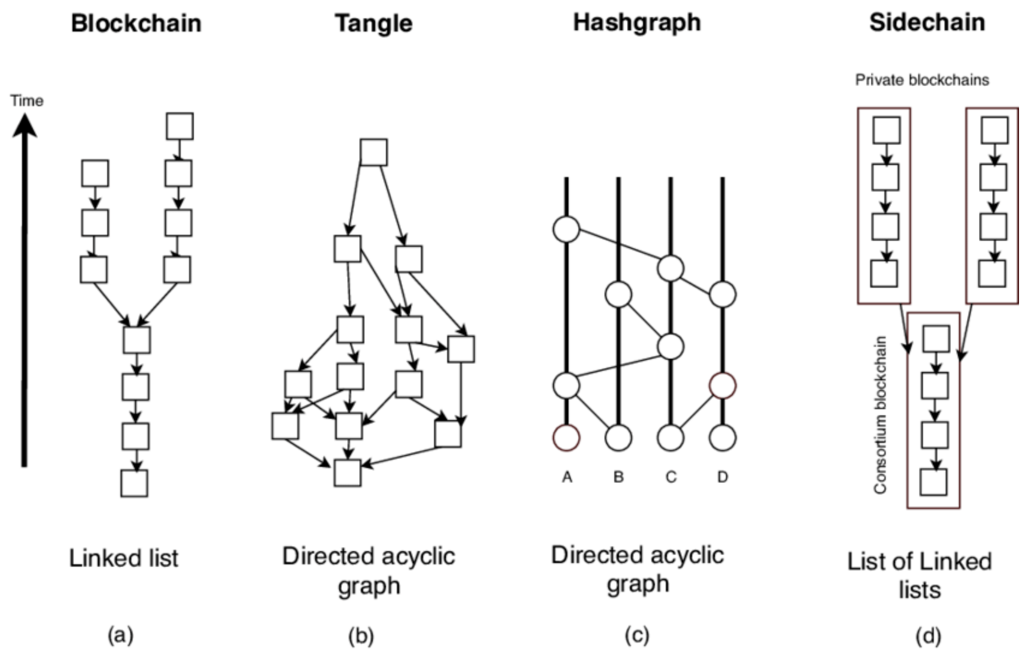

Figure 2.1: An overview of the existing DLTs
[EIP18]

2.1.2 Blockchain

Data Centralization is an issue because apart from the word of the organization that holds the data, there is no guaranty of data validity and fidelity. Any entity with it's own personal interests could modify said data for their own benefit. Decentralized data with distributed systems is a solution for this problem.

Blockchain first appeared in 1991 when Stuart Haber and W. Scott Stornetta created a cryptographic chain of blocks to timestamp documents making the system tamper-proof [HS91]. This Concept of chain of blocks can be compared to a Linked List structure, where each block is a node with a link to the previous block. Instead, on a Blockchain, we keep the concept of Node, but they point to the hash of the previous block, via the hash pointer. In each Node exists a *Header* and a *Body*, the first one contains the Hash to the previous block, a Timestamp, a Nonce that is a hash value used in the creation and validation of a block as well as the Merkle Root, which is a binary tree structure, where each leaf contains the hash of a transaction stored in the respective block, while the parent nodes contain concatenation of the child's transactions hash [Lia20]. Figure 2.2 describes the abstract structure of a blockchain, specifically the various components present inside the block.

In 2008, Satoshi Nakamoto introduced Bitcoin into the world [Nak08], changing the financial transfers landscape and using Blockchain as it's base mechanism. Bitcoin is a cryptocurrency that is built on a decentralized P2P network where each actor holds a copy of the whole chain, as well as employing a consensus algorithm called Proof-Of-Work (PoW), making it so that any change to a block on the chain requires 10 minutes to fulfil such action. The nature of the network, the use of PoW as the consensus algorithm, the permission type access, as well as the cryptographic algorithms that support the system, ensure a layer of security that prevents data tempering.

Blockchain, as many other DLTs, can be applied in other contexts such as health or the government

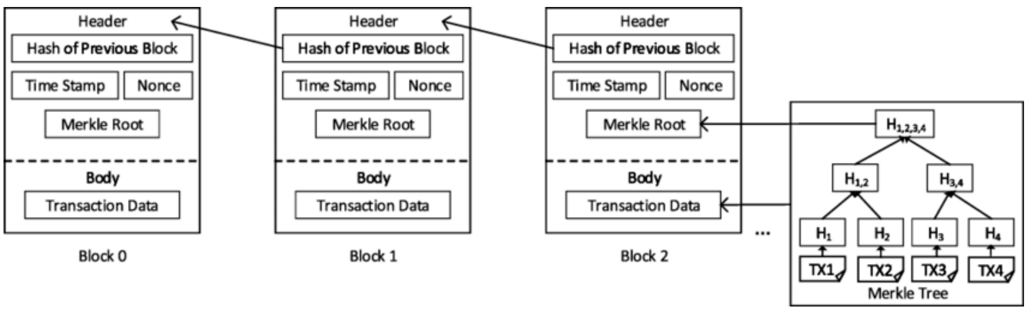

Figure 2.2: The structure of a Blockchain block. [Lia20]

[SZJS22], making it a viable option to solve the existing data security and fidelity in IoT.

Blockchain Architecture

In a blockchain, a block has certain data and metadata, such as timestamps of creation, current block hash and previous hash, and any other necessary and relevant data. The first block of the chain is called the genesis block. It holds data on the chain and is the first created, therefore, it has no previous block hash, as shown in Figure 2.3.

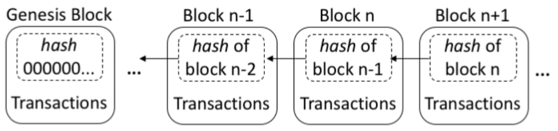

Figure 2.3: Schematic view of the data structure of a Blockchain [TdOCF+19]

Any new transaction is registered as a new block and included into the chain and once that is done it cannot be removed or altered [SP22]. A block will be accepted into the chain if it is considered valid, meaning it has been approved by the implemented consensus algorithm. Since every block has the hash of the previous block, any modified block will have all succeeding blocks recomputed, which can take, in some scenarios, an impossible amount of time to compute.

Consensus Algorithms

A consensus algorithm is a way for participants in a network to reach a mutual agreement on a certain addition or update to the ledger. There are several implementations of consensus algorithms, and various technologies choose to use different algorithms for their specific scenarios and use cases.

There are two big types of blockchain implementation, a Permissioned approach and a Permissionless one. The differences are rather simple; in a Private Blockchain (Permissioned), for an individual or entity to become part of a network and interact with the ledger, they must be authorized to do so. A Permissionless Blockchain is the exact opposite where there is no permission needed to enter or interact with the network. Figure 2.4 portraits the different types of blockchains, as well as describing some of the most used consensus algorithms.

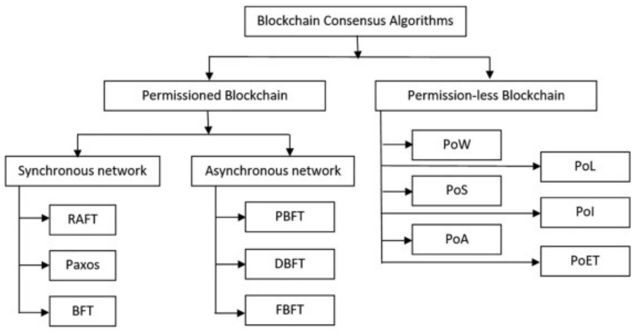

Figure 2.4: Classification of Consensus Algorithms. [SP22]

Permissionless Blockchain

A Permissionless or Public Blockchain is at the centre of the most common and known implementations of Blockchain, which is in various cryptocurrencies. This type of architecture makes for a fully decentralized network across unknown participants which usually produces a larger network and it's counterpart. With this increase in size we get a broader decentralization of the network and more security concerning attacks, but becomes less efficient and harder to scale.

Some examples of public Blockchains are Bitcoin [Nak08] that uses the Proof-Of-Work (PoW) consensus algorithm [CWW$^+$18], ethereum [KMM$^+$18] that has merged and now uses the Proof-Of-Stake algorithm (PoS) [GKR18]. Other examples of permissionless consensus algorithms are Proof-Of-Importance (PoI) [ACAH22], Proof-Of-Activity (PoA) [BMZ18], Proof-Of-Elapsed-Time (PoET) [BMZ18] and Proof-Of-Location (PoL) [Mig18]. These are the most popular and depicted in Figure 2.4, but these aren't the only ones that exist. There are several proposed consensus algorithms that could be adapted and implemented, as various articles propose to alter or update the algorithms enumerated previously.

Permissioned Blockchain

A Permissioned Blockchain is partially decentralized since it's distributed across known organizations. On contrary to a public Blockchain, this type of network is more robust with transactions information since ledger interactions are based on permissions and it's also faster and easier to scale. The problems arise with the limited decentralization potential, making the network easier to prone tampering from the organizations owners and their personal agendas.

One of the biggest concerns with a permissioned Blockchain is a crash, being it a network or byzantine fault. The first one is the most simple to solve, and the big issue lies with byzantine faults. In a distributed system a failing part can create wrong or miss interpreted outcomes that gravely influence the systems performance, as simply explained with the Byzatine Generals Problem [LSP82]. A system that is Byzantine Fault Tolerant is a system where a node's failure or misbehaviour will still be able to continue operating.

As shown in Figure 2.4, we have two major groups, synchronous and asynchronous networks. The first one requires a common time clock which use algorithms like RAFT [HMZ20], Paxos [CAD18] and BFT [QCM$^+$22]. The first two address a network fault crash while the last one focus on Byzantine crashes.

In a real scenario this isn't the ideal path since different time zones can become an unwanted constraint. Some of the consensus algorithms used in this implementation are Pratical BFT (PBFT) [CL02], Delegated BFT (DBFT) [YMA22], and lastly the Fedarated Byzantine agreement (FBA) [IDB18].

2.1.3 The Hyperledger Project

The Linux Foundation, in collaboration with IBM, founded the *Hyperledger Project* as an open-source initiative aimed at fostering a global ecosystem for enterprise-grade blockchain technologies. The project's primary goal is to provide modular, interoperable frameworks that enable organizations to develop distributed ledger solutions suited to their operational and regulatory requirements. Unlike public blockchain platforms, Hyperledger projects are typically permissioned, focusing on scalability, privacy, and performance rather than cryptocurrency or token economies.

Among the various frameworks under the Hyperledger umbrella, **Hyperledger Fabric** and **Hyperledger Indy** stand out as leading implementations of permissioned networks. Both frameworks support decentralized trust across known entities, but they address distinct application domains: Fabric serves as a general-purpose distributed ledger platform, whereas Indy focuses on decentralized identity management.

Hyperledger Fabric Architecture

Hyperledger Fabric (HLF) is a modular and configurable blockchain framework designed to support a broad range of enterprise use cases. Its architecture separates core components such as consensus, membership, and smart-contract execution, allowing system designers to tailor deployments to specific performance, scalability, and security requirements. This modularity makes Fabric particularly relevant for Internet-of-Things (IoT) integrations, where data volumes are high and latency constraints are strict.

Modularity. Fabric's design follows a plug-and-play philosophy. It supports configurable consensus mechanisms, identity-management protocols, cryptographic libraries, and state-database backend. The principal modular components include:

- a pluggable **ordering service**, responsible for establishing transaction order and distributing blocks to peers;
- a pluggable **membership service provider** (MSP), which manages digital identities and cryptographic credentials;
- a peer-to-peer **gossip service** for block dissemination;
- containerized **smart contracts** (chaincode), executable in standard programming languages;
- a configurable **ledger state database**, supporting different DBMS options;
- a customizable **endorsement and validation policy**, which defines transaction approval rules per application.

This modularity distinguishes Fabric from monolithic public blockchains, enabling independent evolution of each component. For instance, organizations can select a state database or consensus protocol that best aligns with their operational context.

Governance and Identity Management. Fabric embeds a governance framework centred on the MSP, which ensures accountability and secure identity lifecycle management. Each organization within the network maintains its own MSP, issuing X.509 certificates through a Certificate Authority (CA) and managing roles and permissions. Unlike public blockchains such as Bitcoin or Ethereum, where participants are anonymous and governance emerges through open consensus, Fabric defines governance explicitly, enabling endorsement policies that reflect real-world business rules. Additionally, Fabric supports multi-channel architectures, where organizations transact privately within dedicated channels while sharing the same network infrastructure.

Ledger State Databases. Hyperledger Fabric supports two main database options for maintaining the world state:

- **LevelDB**, the default embedded key–value store, optimized for high-throughput key-based lookups;
- **CouchDB**, an external JSON document store that supports rich queries via JSON selectors and secondary indexes, well-suited for applications requiring multi-attribute filtering or temporal range queries.

The choice between the two represents a trade-off between performance and query flexibility, with LevelDB providing faster write operations and CouchDB offering enhanced analytical capabilities.

Ordering Service and Consensus. The ordering service forms the backbone of transaction synchronization across the network. It establishes a total transaction order and assembles blocks that are then distributed to peers for validation and commitment. Modern Fabric versions implement the **Raft** consensus algorithm, a crash-fault-tolerant protocol that balances simplicity and reliability. Earlier mechanisms such as Solo and Kafka have been deprecated in favour of Raft, which natively integrates into the orderer nodes and allows the definition of parameters such as *BatchSize* (number of transactions per block) and *BatchTimeout* (maximum waiting time before block creation).

Transaction Flow. Fabric follows an *execute–order–validate* model:

1. **Execute**: clients submit transactions that are simulated and endorsed by peers according to chaincode logic;
2. **Order**: endorsed transactions are ordered into blocks by the ordering service;
3. **Validate**: peers validate the ordering and endorsement policy before committing blocks to the ledger.

This model differs from the *order–execute* design of traditional blockchains, reducing nondeterminism and improving concurrency.

Fabric is implemented in the Go programming language and uses the *gRPC* protocol for communication among peers, orderers, and clients. Its robustness, combined with modular design and fine-grained access control, makes it one of the most widely adopted permissioned blockchain frameworks. It also serves as a foundation for numerous *Blockchain-as-a-Service* (BaaS) offerings [SZA$^+$22], enabling organizations to deploy decentralized solutions with minimal development effort.

Hyperledger Indy. Hyperledger Indy, another framework within the Hyperledger ecosystem, provides decentralized identity management services. While it shares Fabric's permissioned design principles, its focus is on verifiable credentials and self-sovereign identity. Indy relies on the **Plenum** consensus mechanism, a variant of Redundant Byzantine Fault Tolerance (RBFT) [AMQ13], optimized for identity transactions. Each batch of credential operations undergoes a three-phase commit process, ensuring consistency and resistance to malicious actors.

Together, Fabric and Indy demonstrate Hyperledger's commitment to modular, open-source, and enterprise-ready distributed ledger technologies capable of supporting diverse domains, from financial systems to IoT networks.

2.1.4 Internet Of Things (IoT)

IoT revolutionized the way society goes about its daily tasks. Smart homes [ZCDV17], smart cities, health monitoring and improvement technologies, smart industries and so many other quality of life transformations are enabled through IoT [KTZ19].

The massive adoption of IoT further increased the existing dispersion of data through multiple systems. Having devices constantly connected to a network, streaming data, enhances the need for centralized platforms, tackling issues from business development and management to simple day-to-day activities and habits. Although there are many applications and development in this area, it still has a lot of investigation to be made.

Furthermore, one of the problems with centralized platforms are how the manufacturers position their IoT solutions. With the evolution of systems to cloud native, a lot of manufacturers decided to push for a Software-as-a-Service (SaaS) model, where the proprietary devices connect to their IoT service, whilst exposing the device's data in a seamlessly and fully integrated experience. In paper this is a great solution, and in limited and highly controlled ecosystems like cloud Providers. However, in the IoT landscape there are a lot manufacturers, industry standards and differing scope applications, which makes open architectures much more interesting. As with any piece of software there are pros and cons of using proprietary or open-source platforms, and as with most cases the size of the corporation, performance expected, available budget and business goals are what define the type of platform used.

Regarding development, one of the biggest issues associated with IoT is the performance and scalability of a system. Most devices will be either mobile devices or integrated systems, where the processing power is lacking, battery life can also be an issue, and having full fledged implementations will be mostly impossible [Gur21].

Architecture

As explained in the previously there are various approaches to build and develop IoT platforms and devices. For this reason it's usual for multiple devices to integrate with different platforms, with their respective integrations. This is a resource consuming process as new developments are always overhead cost of the platform owner.

In the attempt to create more consistency between systems, information and communications technology (ICT) standards development organizations have been trying to create a set of technical specifications for IoT systems. The most relevant are the IEEE Standard for an Architectural Framework for the Internet of Things [IEE20], the ITU-T Y.2060 Recommendation [Int12], the oneM2M standards [one24], and lastly a framework for service oriented architecture (SOA) [GNLF+22]. There are also various platforms which are built on top of these standards such as the OpenMTC platform, an open-source reference implementation based on the oneM2M standards [ECMB14].

Figure 2.5 is a representation of a fully fledged IoT architecture, composed from various iterations and evolutions of state of the art IoT platforms. This architecture contains the same application-middleware-devices business logic as presented in [JUM+16], whilst also integrating a security and management block. Every other block is also further specified, without limiting its application, and are bellow described based on the definitions provided in [DBCB+22].

1. **Sensors and Actuators**: These physical devices constantly monitor the environment in which their set into. Actuators, however, respond to IoT requests and can also perform actions such as open

a light switch, adjust the temperature or even remotely close the trunk of the car. One of the biggest issues regarding IoT sensors is their lack of native resource capability, they usually have light processors coupled with minimal storage. Furthermore, with the massive evolution of Artificial Intelligence, devices have grown in their capability of edge computing for preprocessing, filtering or anomaly detection.

2. **Communications Network**: This layer is responsible for ensuring connectivity between every component inside the network. It defines the how the communication is physically made, the respective protocols and communication flow. There are various protocols used for internet access, such as *IPv4*, and *IPv6* or *6LoWPAN*, the IPv6 over Low-Power Wireless personal Area Networks, an adaptation for low power devices, solving the scalability issues present in IPv4.The technology used will vary from the range and intended data rate.

3. **IoT Gateway**: Acting as a middleware bridge between the platform and the IoT devices, this layer has the objective of normalizing communication network protocols and data formats. If the network needs to scale, multiple gateways should be implemented to address different data stream sources.

4. **IoT Platform**: The platform is the heart of the ecosystem. It's where the business logic will be defined, as well as all the data is stored. It directly connects the devices to the applications their data will serve once fully processed. It's composed of three foundational blocks:

   (a) **IoT Platform Management** is the block that defines the management functionality of the platform itself. It handles device onboarding, authentication and configuration inside the network.The platform must also provide a unified interface that allows devices using different protocols the same experience and performance. Also monitors and controls the status of the network and respective devices.

   (b) **Data Storage and Analytics** is responsible for storing sensor data in the desired format (relational, NoSQL, time-series, others), with a wide array of analytical tools to be used. The lifetime of the data must be predefined, as there will be collections used for historical records and statistical models, whilst others are suited for immediate usage, such as temperature spike alerts in industry equipments.

   (c) **Application Enablement** is the block that defines how the platform connects to real world applications. It will expose APIs, SDKs and other integration frameworks that can develop IoT applications. The idea behind it is to define the base rules and ground limits of how applications interact with the available functions inside the platform.

5. **Applications**: The applications are the real world business implementations of the use case they were developed to support. They can be dashboards that visualize data, HVAC controllers, remote car starters or any other business application. They can be implemented as cloud-native, mobile apps or embedded into a web service, the main idea is that they need to consume that from the IoT platform, be it for streaming data regarding room temperature, or predictive data of how our electrical consumption will vary for the next month.

6. **Security**: This block must ensure compliance of regulations, data confidentiality, integrity and availability. Defines with what protocols users authenticate, ensures the management of certificates and keys, as well as monitoring security events.

7. **Management**: This is the brains of the ecosystem, it will intelligently orchestrate tasks inside the whole system, ensuring fault management, system performance and successive updates and configurations.

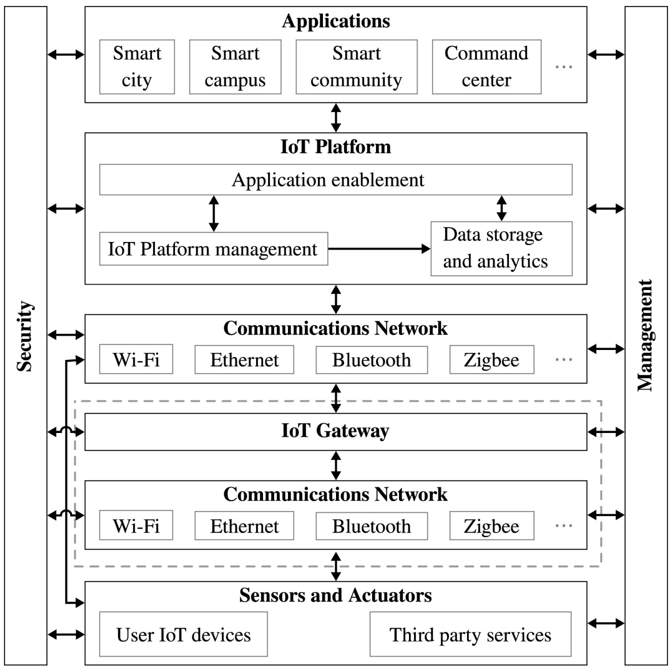

Figure 2.5: IoT architecture. [DBCB+22]

Internet of Things (IoT) and Blockchain Integration

Distributed Ledger systems naturally have a high interest in regards to IoT systems. Decentralizing data in a system can be a requirement or purely a positive advantage, in which IoT devices can exchange information over the internet through a secure network. Data can be easily traced, thanks to its immutability, and any confidence concerns are automatically removed from the equation. With DLTs IoT has the possibility to be completely revolutionized in many crucial areas where data integrity is key, such as health, industries, businesses, finance, and many other areas [MMA23].

IoT can benefit greatly from DLTs, such as Blockchain, on various areas like the following:

- Decentralization: Blockchain has the intrinsic property of decentralizing information with its peer-to-peer network. It also improves against fault tolerance, like the Byzantine Fault, as well as help with IoT scalability;
- Identity and Autonomy: With a standard Blockchain network, participants (users) can be easily identified and its transactions easily traced. IoT systems will also have the capability of interacting without the need for servers;
- Data Authenticity and Security: Every piece of data that was transacted, can be traced back to their original state, validated by smart contracts. Current existing protocols in IoT can greatly benefit from this.

2.2 Related Work

Blockchain and IoT have been proposed to address multiple challenges regarding security, scalability, and performance. One of the most relevant use cases is in the financial services industry, where blockchain is used to conduct peer-to-peer transactions, without the need for an intermediary. Healthcare, a highly regulated industry, is also leveraging this technology to secure data storage and private sharing with patients. The following section describes different representative works from the literature, allowing for a comprehensive understanding of how these systems integrate, and the associated results.

Blockchain Integration in IoT: Applications, Opportunities, and Challenges

Gholami et al. (2025) present a comprehensive survey of blockchain-IoT integration, examining application domains, security and privacy requirements, as well as the primary scalability and interoperability challenges emerging when deploying distributed ledger technologies in heterogeneous IoT environments [GGD^{+}25]. The authors classify recent work according to use-cases (e.g., smart grid, supply chain, healthcare), identify the decentralisation, anonymity, and device-constraint features of blockchain that do benefit IoT, but then highlight obstacles such as resource limitations, network connectivity, and the absence of standardised integration frameworks. A forward-looking section outlines open research directions including lightweight consensus, cross-chain interoperability and trustworthy data sharing. While not an empirical study, this survey ties directly into this thesis by framing the landscape of IoT-blockchain deployments: the identified gaps (resource-constraint handling, consensus overhead, integration frameworks) mirror those addressed in our custom Hyperledger Fabric benchmarking network with IoT-specific data structures and the transition from LevelDB to CouchDB state database.

Performance Enhancement in Blockchain-based IoT Data Sharing Using Lightweight Consensus Algorithm

Haque et al. (2024) present a blockchain framework tailored for IoT data sharing, addressing the scalability and latency challenges inherent in typical ledger systems [HAA^{+}24]. The authors implement a hybrid design that combines the Delegated Proof of Stake (DPoS) consensus mechanism with sharding and off-chain storage via IPFS. Empirical evaluation shows throughput rising with load, e.g., at 500 TPS achieving 11.094 ms latency, and only slight degradation at 2000 TPS (11.205 ms) under constrained IoT-node settings. They also analyse trade-offs among throughput, latency and up/down-stream times. Importantly, the architecture is designed for resource-constrained IoT devices and demonstrates superior performance compared to edge-computing–based schemes. This work directly connects to our research by offering a pragmatic IoT-friendly blockchain architecture; its consensus-sharding-storage triad provides a conceptual benchmark for our custom Hyperledger Fabric & IoT network, particularly when assessing state-database type (LevelDB vs CouchDB) and exploring performance boundaries in distributed ledger deployment for IoT scenarios.

Blockchain–IoT Integration for Industry 5.0: Platform Selection and Architectural Considerations

Sizan et al. (2025) survey blockchain–IoT integration through the lens of Industry 5.0 and propose a framework to guide the selection of suitable blockchain platforms for IoT deployments [SDL^{+}25]. The review contrasts architectural options (e.g., on-chain/off-chain data handling, DAGs vs. blockchains) and synthesizes requirements around security, interoperability, and scalability for heterogeneous device environments. The authors emphasise lightweight consensus and edge/near-data processing to mitigate latency and resource constraints, and outline research directions on standardised integration and trustworthy data sharing. This directly informs our Fabric-based design choices by reinforcing modular architecture and by motivating our evaluation of performance under IoT-oriented constraints (e.g., varying state-database backends and endorsement policies) using Hyperledger Caliper.

Performance of Private Permissiond Blockchains

The authors conducted a series of tests to evaluate the performance of two permissioned frameworks, Parity and Multichain [TdOCF$^+$19]. The first is used for developing solutions based on the Ethereum network, whilst Multichain is based on the official client of the Bitcoin network, the Bitcoin Core. In experimental setup, they attempted to simulate a realistic workload with multiple transactions between clients over the course of one hour. They measured metrics such as transaction time, transaction seek, elapsed time to find a transaction, or block seek, elapsed time to find a block identifier. Their results emphasize the possible issues surrounding scalability with private blockchains and IoT.

Blockchain Bottleneck Analysis Based on Performance Metrics Causality

Song et al. (2024) propose a causality-based framework to identify performance bottlenecks in blockchain systems [SZL$^+$24]. The authors introduce eighteen performance metrics distributed across contract, network, data, consensus, and system layers, analysing their causal relationships under increasing workloads. Their platform-agnostic measurement system, tested on ChainMaker, Ethereum, and FISCO BCOS, revealed that as the system approaches saturation, the causal dependencies among metrics weaken, signalling emerging bottlenecks. Despite not focusing on IoT scenarios, this study provides a valuable methodology for diagnosing performance degradation in distributed ledgers. The presented metric taxonomy and causality analysis complement this work's benchmarking approach with Hyperledger Caliper, offering a multi-layered perspective that can be adapted to assess IoT-oriented blockchain architectures such as the one implemented in this dissertation.

Hyperledger Fabric Distributed Computing Performance Evaluation

Androulaki et al. (2018) described the architecture of Fabric as the "first blockchain system that runs distributed applications written in standard, general-purpose programming languages, without systemic dependency on a native cryptocurrency" [ABB$^+$18]. Their experiments focused on transaction performance, concurrency handling with up to 100 clients and state database options. In 2018 there were no benchmarking frameworks for Fabric, or blockchain in general, they resorted to the use of the *Fabric Coin*, based on the Bitcoin Data Model. Furthermore, they define a standard methodology where they would populate the network with assets, coins, and increase the number of clients until network saturation, lastly they'd run transactions between all the clients. The results focused demonstrated the impact of several configurations in the networks performance, highlighting the block size as one of the most important parameters.

Hyperledger Caliper: Performance Benchmark Framework for Blockchain Systems

The Hyperledger Performance and Scale Working Group introduced Hyperledger Caliper, the first modular benchmarking framework developed within the Hyperledger ecosystem to evaluate blockchain performance under standardized workloads [PG]. The framework defines a unified interface for integrating different blockchain platforms, including Fabric, Sawtooth, and Ethereum, and reports metrics such as throughput, latency, resource consumption, and success rate. Its extensible adapter layer allows testers to design custom benchmarks and workloads while maintaining reproducibility across platforms. The authors demonstrate Caliper's capability through multi-network experiments, highlighting that configuration parameters, such as block size, transaction rate, and endorsement policy, strongly influence system behaviour. This work directly supports the empirical foundation of the present dissertation, as the same framework was employed to benchmark the proposed Hyperledger Fabric & IoT implementation. The methodology, metric definitions, and workload abstraction mechanisms described by the group, ensure the comparability and methodological rigour of the performance evaluations presented in Chapter 4.

Framework for Blockchain and IoMT integration with Federate Learning in Healthcare

Bhasker et al. (2025) proposed the usage blockchain combined with the Internet of Medical Things IoMT to address data security, privacy concerns and compliance in traditional systems based on IoMT [BRS+25]. The authors took it a step further by adding federated learning to this ecosystem for real-time health monitoring, ensuring the base premises of security and compliance. They simulated an ICU scenario with 100 IoMT devices, wearables and sensors, and anonymized the health data collected. The metrics were used data privacy, intrusion detection efficiency, disease detection accuracy, proactive healthcare management and interoperability. The results were 98.73%, 97.16%, 96.42%, 98.37% and 96.74%, respectively. The study demonstrates that by integrating federate learning alongside blockchain and IoMT can boost the systems security and resilience.

2.3 Summary

This chapter presented the generalized theoretical foundations that together construct the basis of our work, on how Blockchain and Internet of Things systems can be integrated. We began with an overview of Distributed Ledger Technologies, the foundation of Blockchain, and how decentralized networks are used to record and validate data across multiple participants without a central authority. Blockchain is the most used and studied implementation, ensuring immutability, traceability and trust via consensus mechanisms. We also explored different types of networks, Permissioned or Permission-less, which will greatly impact the performance and behaviour.

Furthermore, we discussed alternative DLT implementations such as Tangle, Hashgraph, Sidechain and Holochain. These approaches are defined by their different implementations of the data structures, consensus algorithms and other specifically imposed rules

There are various technologies where we can build Blockchain solutions, but since we wanted a fully modular, enterprise grade network, we opted for the Hyperledger ecosystem. We looked at various products such as Indy, a focused Blockchain for decentralized identity applications and Fabric, which is a Permissioned network where smart contracts define the business logic. Although Indy can be used as a general purpose Blockchain or Ledger, modularity was of big importance, therefore we chose Fabric. We also outlined Fabric's architecture and some of its characteristics.

We described the current landscape of the IoT landscape, how it started and how it is currently evolving. Focused on the fact that these devices generate vast amounts of data, whilst facing multiple challenges regarding authenticity and fault tolerance. This chapter also discusses how Blockchain can address many of these problems.

Lastly, this chapter highlights some relevant literature related to blockchain and IoT, their approaches, methodologies, and results.

3

The Proposed System

The literature review in Chapter 2 portrayed a clear landscape of the current challenges when integrating IoT with blockchain. Building on this, our goal is to evaluate a blockchain-based IoT architecture capable of handling large sensor data accurately whilst ensuring data trustworthiness throughout the whole data lifecycle.

Chapter 3 describes a set of specifications, designs, and rules that define our system. Our proposal highlights a high level design (HDL) with multiple systems integrated, constructing a modular approach to a blockchain architecture. The system developed demonstrates how a permissioned blockchain network can be the infrastructure for IoT data storage, traceability and integrity.

The technology used was Hyperledger Fabric, a permissioned blockchain from the Hyperledger ecosystem, which is characterized by its modular architecture and enterprise-grade performance and capacity. It was configured with two peers, simulating different organization and an orderer, which orchestrates and ensures the validity of every committed transaction. With this configuration, we were able to emulate an industrial IoT use case. The blockchain has an underlying ledger, which stores the sensors data through business logic conducted in the smart contracts implemented, written in GO.

Furthermore, in the attempt to simulate real-world IoT interactions, we developed a custom client application using the Fabric Gateway SDK, which enabled controlled transaction submission and data querying. This client was designed to emulate IoT devices sending structured data payloads to the blockchain network. These architectural choices and their influence on performance are systematically discussed in the sections following.

Data was stored in the ledger through one of the choice state databases, LevelDB or CouchDB, which we will discuss further in this chapter. The chaincode functions created were designed to handle common IoT operations: Read, Write and Update. Some examples of such operations include the addition of new sensor reading, consulting the ledger for intervals of readings, or updating sensors network data.

The implementation also incorporates an integration layer for analytics and monitoring, preparing the system for future extensions such as on-chain telemetry visualization or off-chain data processing using Prometheus, Grafana, or Artificial Intelligence (AI) pipelines. The primary contribution of our work is the end-to-end design and development of a blockchain-enabled IoT data management system capable of supporting continuous, verifiable information exchange without centralized control.

3.1 The Proposed Goals

The integration of Blockchain technologies within the Internet of Things (IoT) environments presents a unique set of challenges and opportunities. This dissertation proposes a system architecture that leverages a Permissioned blockchain, Hyperledger Fabric, to address key concerns related to data integrity, traceability, and decentralization in distributed sensor networks.

The primary goal of this work is to evaluate whether a blockchain-based infrastructure using Hyperledger Fabric can effectively support high-frequency, high-volume IoT workloads while maintaining performance and ensuring trust in the recorded data. To achieve this, we formulate the following specific objectives:

- **Design and implement an IoT-oriented blockchain architecture:** Develop a modular system that simulates the interaction of IoT devices with a blockchain ledger, focusing on data traceability, structured payload ingestion, and decentralized validation of records.

- **Explore different data modelling approaches:** Analyse the performance implications of various ledger-level data structures tailored for IoT sensor data. This includes both append-based models and key-based timestamped entries.

- **Evaluate the impact of network configurations:** Compare the performance of different state databases, *LevelDB* and *CouchDB*, Ordering Service, Membership and Identity Management (MSP), Endorsement Policies or Block and Transaction Parameters in regards of throughput, latency, and scalability.

- **Leverage benchmarking frameworks and custom workloads:** Employ Hyperledger Caliper for standardized benchmarking and supplement it with custom test scripts that emulate realistic IoT usage patterns and load.

- **Assess network scalability and client concurrency:** Test the system under various conditions of client parallelism and transaction throughput to evaluate how Hyperledger Fabric handles horizontal scaling in IoT-like scenarios.

- **Prepare the foundation for analytics and visualization components:** Although not the primary focus, this work also proposes a system design extensible to include modules for monitoring, data visualization, and off-chain analytics using external platforms such as Prometheus, Grafana, or AI models.

Together, these goals aim to demonstrate the feasibility of a blockchain-based infrastructure that can sustain the operational requirements of IoT environments, and provide a foundation for future work in performance optimization, real-time analytics, and decentralized identity.

3.2 Specification and Design

This section describes the logical design of the proposed system, establishing the conceptual foundation on which it was built. The developed solution aims to ensure secure exchange and storage of data through a Permissioned blockchain network, without compromising scalability or performance.

3.2.1 System Requirements

The proposed system was designed with a set of requirements in mind. Firstly, it needed to be a real alternative to the existing data storage ecosystems, where raw data is filtered, and stored according to certain business rules. Furthermore, it needed to allow for the most basic operations such as retrieve specific information for the database, in this case the ledger, write and remove data. Lastly, it needed to do all the above without losing performance or forming a silo where further developments became close to impossible. This section defines the general guidelines of the designed system.

Functional Requirements

- IoT devices need to be able to register sensor readings on a blockchain in real time or batch mode
- Data stored in the ledger must be immutable and traceable through cryptographic validation and consensus mechanisms
- Network participants must be able to retrieve current and historical data from the ledger
- Support interaction between multiple organizations, where peers validate transactions under defined policies, such as the defined consensus algorithm
- The system should be exposed through a standard interface which allows external applications to interact and write or retrieve data

Non-Functional Requirements

- **Scalability**: The system needs to be able to accommodate large volumes of input data
- **Integrity and Security**: Every data entry must be authentic and tamper-resistant
- **Performance**: Operations performed on the system should be competitive, when compared to the traditional data storage systems
- **Interoperability and Extensibility**: The proposed architecture should be modular and compatible with Cloud or On-Premises environments. It should also allow for easy integration with different IoT frameworks, as well as future extensions attempting to implement business intelligence solutions off the network.

3.2.2 Data and Interaction Model

The proposed system models sensor data as digital assets stored on the ledger. Each data point contains metadata that identifies the source device and contextual information that describes the respective measurement. Figure 3.1 depicts a typical IoT data model.

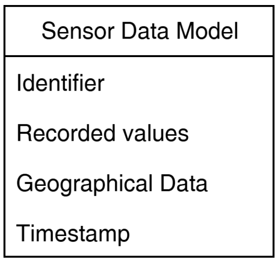

Figure 3.1: IoT Sensor Data Model

In the proposed system, transactions represent actions that interact with the ledger, modifying or retrieving the state of these assets. The main transaction types and operations are the following:

- **CreateSensor**: Create a Sensor (asset) in the ledger
- **ReadSensor**: Read values for a Sensor
- **UpdateSensorReading**: Update values for an existing sensor
- **DeleteSensor**: Delete an existing sensor. Since the blockchain is immutable, assets can't be deleted from the ledger, only flagged in the in the world state database as deleted.
- **getIntervalReadings**: Read all the readings for one or multiple sensors within a certain time interval (1 hour, 1 day, 1 week, etc.)
- **getSensorHistory**: Read every transaction and state change from a sensor

In Figure 3.2 we describe a simple interaction between an IoT Sensor, which is also a data producer, a client application and the blockchain ledger. In this draft, the sensor updates an existing asset with a new reading, which goes through a gateway, that transmits the updated data request into a transaction for the ledger. The transaction must be validated and endorsed to be written into the blockchain. At the same time, an external client is also interacting with the gateway, in an attempt to retrieve interval readings. The request is once more converted to a transaction, and after proper validations, the payload is sent with the desired data.

System Behaviour and Logical Flow

The system follows a sequential logic that governs the lifecycle of sensor data:

1. **Data generation**: Sensors or simulated clients periodically produce structured readings.
2. **Transaction submission**: Each reading is formatted as a blockchain transaction and digitally signed.
3. **Validation and ordering**: Peers verify the transaction validity, and the orderer sequences and commits it to the ledger.
4. **State update**: The state database is updated, maintaining both the current snapshot and a full historical record.

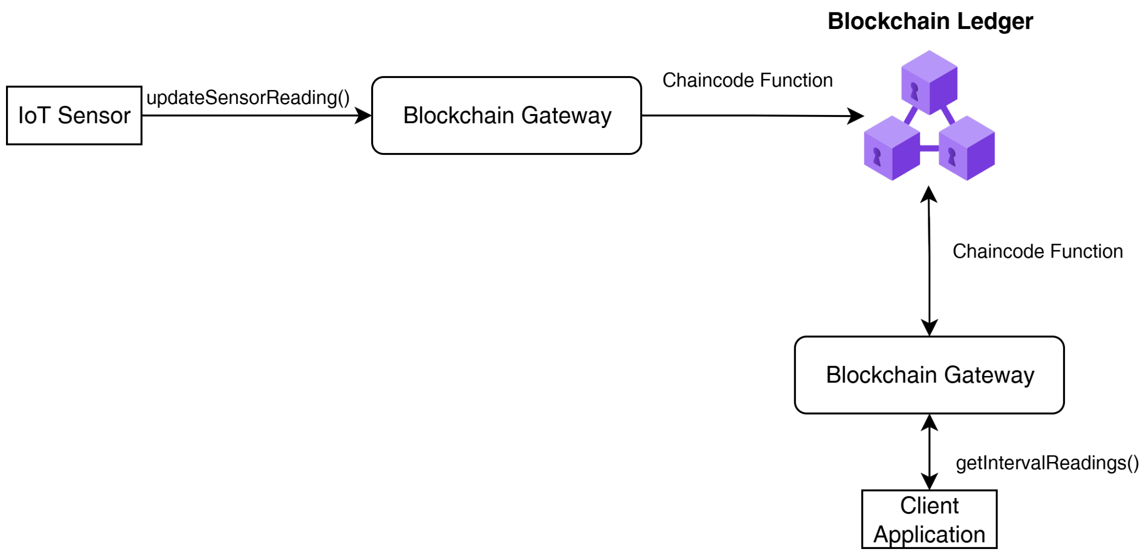

Figure 3.2: Interaction model of the blockchain ledger, an IoT Sensor and a Client Application

5. **Query and analysis**: Authorized entities can retrieve or analyse data through queries or off-chain analytic modules.

This logical process ensures that all operations, from data ingestion to query, are verifiable and consistent across the distributed system.

3.2.3 Design Rationale and Balance

Every design decision adopted was based from common IoT ecosystems, where decentralization and efficiency are key factors for a performant solution.

We chose a Permissioned blockchain to ensure controlled access and regulatory compliance, which perfectly aligns with realistic IoT systems. Furthermore, Hyperledger Fabric offers a highly modular framework, allowing our design to incorporate most external services, whilst ensuring security and validity.

In regards to the proposed data model, we needed to guarantee that historical data was easily accessible and that new readings from concurrent sensors didn't induce overhead into the system. By designing a system of one sensor as an asset, we managed to fully separate every producer into a individual node, with the respective ramifications of previous readings.

Lastly, the high level design, presented in Figure 3.3, was specifically designed for state of the art systems, where there can be multiple types of Cloud systems interacting, and monitoring and analytics modules are ready to be implemented.

3.3 Architecture

This Section presents the architectural design of the proposed system, integrating an IoT network with a blockchain-based distributed ledger using Hyperledger Fabric. The aim is to establish a framework that can handle high volumes of data processing, whilst ensuring its security, integrity and traceability through the whole lifecycle. The following subsections describe the various layers, configurations and mechanisms that compose the system.

3.3.1 Propose System High Level Design

The proposed architecture is designed to address the key requirements of IoT data systems, namely: high throughput, tamper-resistance, decentralized trust, and efficient query capability. The core of the solution is built on *Hyperledger Fabric*.

The architectural components are as illustrated and discussed, including:

- **Fog/Edge Layer of Sensor Devices**: Simulated IoT sensors or devices generating data;
- **ETL Layer**: Filter raw Sensor data; Be able to have a curated data layer that only stores specific data in the blockchain. Similar to a Data warehouse Bronze, Silver and Gold quality logic.
- **Hyperledger Fabric Network**: An instance of HLF with specific chaincode deployed; Instantiates the ledger and respective databases.
- **Fabric Gateway Rest Adapter**: Server running a REST wrapper of the Fabric Gateway SDK; Exposes blockchain interaction via REST API to the exterior.
- **Analytics**: Analytics module for future work (LLMs, ML, etc); A novel approach to the use of blockchain with advanced analytical modules, abstract definition.
- **Visualization & Monitoring**: Grafana & Prometheus for on-chain data visualization as well as network resources monitoring.

The architectural interaction begins at the Fog/Edge layer, where IoT sensor devices generate raw telemetry that is ingested into the ETL pipeline. This pipeline applies a multi-stage curation process inspired by the Bronze/Silver/Gold model: raw measurements are first collected in their unprocessed form, then cleansed and normalised, and finally transformed into a high-quality, policy-compliant subset suitable for on-chain storage. Only the curated Gold-layer records are submitted to the *Hyperledger Fabric* network, where they are encapsulated as blockchain transactions, endorsed by peers, ordered, and committed to the ledger and world state database. Interaction with the blockchain is mediated through the *Fabric Gateway REST Adapter*, which exposes chaincode functions as standard HTTP/JSON endpoints, thereby decoupling external applications from the internal Fabric SDK. Both the blockchain data (via authenticated queries) and off-chain ETL data streams feed into the *Analytics* layer, enabling the development of advanced machine learning and large language model modules that leverage trusted, provenance-guaranteed inputs. Finally, operational and business-level observability is ensured through the *Visualisation and Monitoring* components, where Prometheus captures network and node metrics and Grafana renders both system-level telemetry and on-chain sensor records, providing a comprehensive view of system behaviour and data evolution across all layers.

Since we couldn't replicate data streaming from a set of IoT devices on a Network, we've managed an adaptation with representative batch files. Figure 3.3 illustrates our approach to a fully modular implementation of IoT with blockchain for on-chain traceability and querying, whilst also being prepared for off-chain analytics.

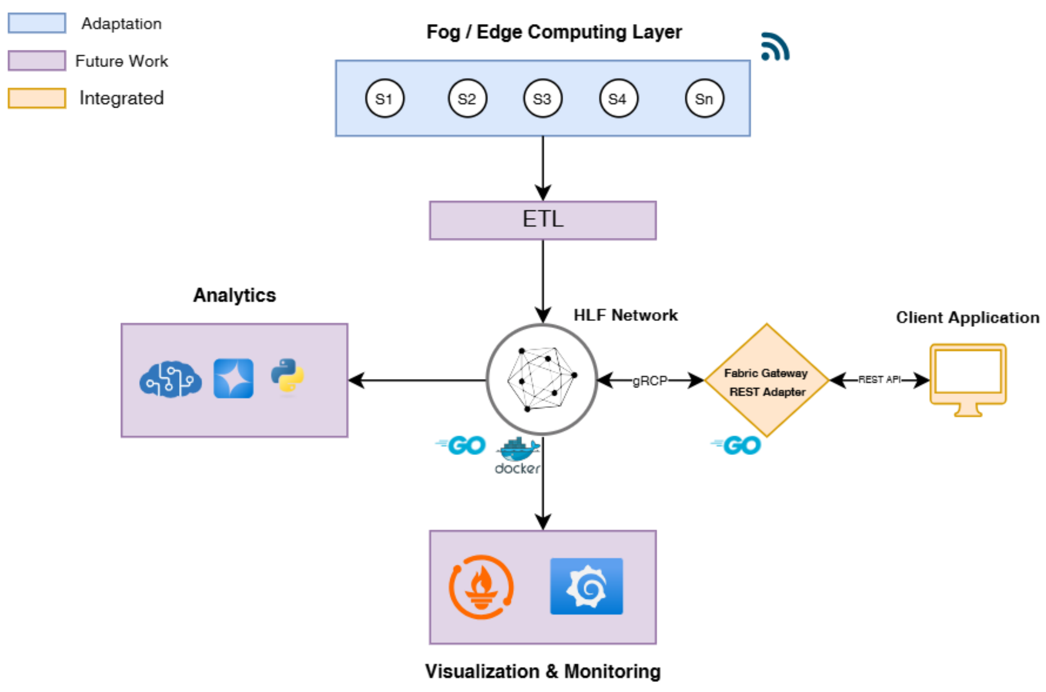

Figure 3.3: High Level Design of the integration of IoT with Hyperledger Fabric

3.4 System Implementation

Section 3.4 highlights the foundations behind the system implementation, its composition and configuration. Our goal was to develop a system capable of handling multiple high volume data transfers, without losing significant performance. As previously explained, modularity, security and traceability are at the core of the system. The following Subsections describe how the system was implemented.

3.4.1 System Environment

The implementation of the proposed system occurred under a controlled environment. The entire infrastructure was deployed on a single host machine running Ubuntu 22.04.5 LTS, using Docker Compose (v1.29.2) for container orchestration and network virtualization.

The processor used was an AMD Ryzen 3350H, with 4 cores and 8 threads, 16GB of DDR4 RAM and a 512 GB SSD.

The deployed Hyperledger Fabric network was based on version 2.5.9, running the same version for the peers and orderer nodes. For the cryptographic material, we could have used Cryptogen or Certificate Authority (CA), and since we stayed in a controlled test environment, we chose Cryptogen since it simplifies identity and MSP provisioning. In an enterprise implementation, most organizations already have a Public Key Infrastructure in place and will use their trusted CA for all their digital certificates, including blockchain.

The network configuration was simple, which allowed us to understand the bare-bone functioning of Fabric. It was composed by three organizations, each with one peer node, and a single ordering service implementing the RAFT consensus protocol. The three nodes were connected into a dedicated channel, ensuring data transactions and endorsement policies where completely encapsulated into their environment.

Lastly, to access performance across different ledger state databases, the system was alternately deployed with LevelDB and CouchDB as state databases. Both configurations were evaluated under identical network and workload conditions to ensure compatibility and comparability.

3.4.2 Network Deployment

The blockchain network was deployed using the Hyperledger Fabric Test Network template, from the official Fabric samples repository. The deployment was fully containerized with Docker Compose, through the network.sh script. Many of the original configurations were maintained, however we changed the Fabric version to 2.5.9 for compatibility issues with some of the tests conducted with Hyperledger Explorer.

The deployment process followed the initialization workflow bellow:

1. **Generation of Cryptographic material through the Cryptogen utility**, defining the certificates and MSPs for each organization
2. **Network instantiation of the docker containers**, Orderer, Peer 1 and Peer 2, and LevelDB/-CouchDB
3. **Channel creation**, where we define a single communication channel, which all the organizations then subscribe to
4. **Anchor peer update**, enabling communications between organizations
5. **Chaincode deployment**, where the respective chaincode is deployed into the channel, with organizations endorsing the process for its completion. This step completes the operational setup of the network.

This configuration reflects the modularity and security of Hyperledger Fabric, where each peer maintains its own ledger and world state, shared through the channel, while the ordering service enforces deterministic transaction ordering. The orderer is responsible for block creation and broadcasting to the network, where then transactions are validated, and finally committed to the ledger.

3.4.3 Chaincode Implementation

Smart contracts are the defining logic behind blockchain applications, and Fabric excels in the modularity and capability to deploy and develop chaincode. We managed to fully emulate many of the CRUD operations that would exist in an IoT system, while also adding specific business logic implementations such as filtering through time intervals, with custom values such as Day, Week, Month or Year. The chaincode was developed using the Go programming language, which was selected for its native integration with Fabric's runtime.

The chaincode defines the data schema, which is structure as an object inside the ledger when stored, as well as the set of transactions which can be used at any time to write, read or update the blockchain. Each asset, is a record, or sensor entry, generated by an IoT device, containing metadata such as a sensorID, a timestamp, and contextual information associated with the business case. The implemented data structure follows a key-value format, where the key represents a unique identifier and the value a JSON representation of the contextual data.

For the key format, we chose a composite key, sensorID:timestamp. By using this method, we are improving our sequential iteration capacity over time-ordered data, whilst ensuring data integrity.

Listing 3.1: Example sensor asset stored on the ledger

```
{
  "sensorID": "sensor_001",
  "timestamp": "2025-03-18T14:52:00Z",
  "temperature": 27.5,
  "location": "Room_A",
  "type": "indoor"
}
```

Smart Contract Functions

The following core functions were implemented to support IoT-like operations within the blockchain network:

- **`CreateSensor(ctx, id, roomID, sensorTimestamp, temp, readingLocation) error`**
  Creates a new sensor reading in the ledger under key `id`. In this system, `id` follows the composite format `sensorID:timestamp` to ensure immutability and correct historical indexing. Fails if the key already exists.

- **`AppendReading(ctx, sensorID, roomID, sensorTimestamp, temp, readingLocation) error`**
  Inserts a *new* time-stamped reading using the enforced key `sensorID:timestamp`. This prevents overwriting. Fails if a reading already exists for that exact pair of `(sensorID, timestamp)`.

- **`ReadSensor(ctx, id) (*SensorData, error)`**
  Retrieves the reading stored at key `id` (*i.e.*, `sensorID:timestamp`). Returns an error if the asset is not found or if deserialization fails.

- **`UpdateSensor(ctx, id, roomID, sensorTimestamp, temp, readingLocation) error`**
  Updates the *value* stored at key `id`. This is intended for corrective metadata updates (e.g., `roomID` or `readingLocation`) and should not be used to shift the timestamp semantically. If a new measurement arrives at a different time, the `AppendReading` function should be used to create a new asset keyed by the new timestamp.

- **`DeleteSensor(ctx, id) error`**
  Removes the current state for key `id` from the world state. The full transaction history remains available through the ledger, ensuring auditability.

- **`GetAllSensors(ctx) ([]SensorData, error)`**
  Performs an open-ended range scan over the chaincode namespace (start=`""`, end=`""`) and returns all readings currently present in the world state. Intended for diagnostics and smaller datasets.

- **`GetHistory(ctx, id) ([]SensorData, error)`**
  Returns the chronological evolution of the asset stored at key `id` by invoking Fabric's `GetHistoryForKey()` API. This function is used for provenance tracking and audit trails.

- **`GetIntervalReadings(ctx, sensorID, startISO, endISO) ([]SensorData, error)`**
  Returns all readings for a given sensor in the inclusive interval `[startISO, endISO]` using a lexicographic key-range scan over composite keys. Internally, the function scans from `sensorID:startISO`

to `sensorID:endISO`. This approach is efficient and database-agnostic, as ordering is derived directly from the key.

- **`GetIntervalReadingsCQ(ctx, sensorID, startISO, endISO) ([]SensorData, error)`**
  Provides the same interval semantics through a CouchDB JSON selector on fields `sensor_id` and `timestamp`, with sorting on both. It requires a secondary index deployed with the chaincode under `META-INF/statedb/couchdb/indexes/sensors.json`:

  Listing 3.2: CouchDB index for GetIntervalReadingsCQ

  ```
  {
    "index": { "fields": ["sensor_id", "timestamp"] },
    "ddoc": "index-sensorid-timestamp",
    "name": "idx_sensorid_timestamp",
    "type": "json"
  }
  ```

  This variant is preferable when combining temporal filters with additional attributes (e.g., room, type).

- **`SensorExists(ctx, id) (bool, error)`**
  Returns whether an asset exists at key `id`. Used by creation and update operations to enforce uniqueness and provide clearer error handling.

To support interval queries efficiently across both LevelDB and CouchDB, the function `GetIntervalReadings` performs a lexicographic scan over composite keys `sensorID:timestamp`, guaranteeing temporal ordering without the need for additional indexes. For CouchDB deployments that require multi-field filtering, `GetIntervalReadingsCQ` leverages a JSON selector backed by an index on `sensor_id` and `timestamp`, enabling expressive queries with predictable performance.

3.4.4 Client Application and REST Gateway

To enable external interaction with the blockchain network, a dedicated client layer was developed using the **Hyperledger Fabric Gateway SDK for Go**. This component served as the primary middleware between external applications and the underlying Fabric network, encapsulating the low-level transaction logic and providing a simplified interface for invoking smart contract functions.

The Fabric Gateway was responsible for managing all stages of the transaction lifecycle, including proposal submission, endorsement collection, and commit confirmation. It effectively abstracted the chaincode interactions into higher-level *transaction calls*, allowing external clients to perform operations such as creating new readings, querying specific sensors, or retrieving historical intervals without direct knowledge of Fabric's internal mechanisms.

In practical terms, the gateway was deployed as a standalone Go-based service and exposed to the outside world through a lightweight **REST API**. Each REST endpoint was mapped to a specific chaincode function, ensuring a clear separation between the blockchain logic and the application layer. The following list illustrates the main correspondence between endpoints and smart contract methods:

- `POST /sensor/create` → invokes `CreateSensor`
- `POST /sensor/append` → invokes `AppendReading`

- `GET /sensor/{id}` → invokes `ReadSensor`
- `GET /sensor/history/{id}` → invokes `GetHistory`
- `GET /sensor/interval` → invokes `GetIntervalReadings`

This modular structure allowed different clients to interact with the blockchain in a standardized manner. To demonstrate this integration, a **Python client application** was defined, simulating an IoT environment where multiple sensors periodically send temperature readings to the REST interface. Each reading was received by the REST server, validated, and then relayed to the Fabric Gateway, which handled the submission of the corresponding transaction to the network.

Each simulated device would generate JSON-formatted payloads containing sensor metadata and timestamped values, and transmit them to the REST service via HTTP requests. This process was done via batch, and not data streaming. The Fabric Gateway, in turn, transformed these incoming requests into blockchain transactions through the SDK, ensuring endorsement and commit procedures were properly executed.

This client–gateway architecture provides an extensible design, where additional front-end or data-processing components can be integrated with minimal adaptation. By abstracting blockchain interactions through RESTful endpoints, the system maintains flexibility for future integration with diverse IoT frameworks or analytics platforms, without requiring direct modification of the underlying Fabric logic.

3.4.5 System Implementation Summary

The implemented system combined a Hyperledger Fabric network with a Go-based smart contract and gateway layer to manage IoT sensor data in a secure and traceable manner. Through Dockerized deployment, composite key modelling, and REST-based client access, the architecture effectively demonstrated a practical and extensible integration between blockchain and IoT environments.

3.5 Summary

This chapter presented the complete design and implementation of the proposed blockchain–IoT integration framework. It began by outlining the conceptual goals of the system and the motivation behind adopting a permissioned distributed ledger for IoT data management. The design specification defined the abstract requirements of the solution, highlighting the need for scalability, data integrity, and efficient querying of time-series information.

The system architecture introduced the logical and physical components that compose the network, including IoT devices, gateway nodes, and the blockchain infrastructure. Particular attention was given to the role of the Hyperledger Fabric framework, whose modular and permissioned model allows the separation of concerns between consensus, identity management, and application logic.

Following the architectural overview, the implementation section detailed the practical realization of the system. It described the deployment environment, the setup of the three-organization Fabric network under the RAFT consensus protocol, and the chaincode developed in Go to manage sensor readings using composite keys (`sensorID:timestamp`). The client-side integration was implemented through the Fabric Gateway SDK, exposing REST endpoints that could be consumed by external applications, such as a conceptual Python IoT client.

4 Experimental Results

This Chapter presents the experimental evaluation of the proposed blockchain–IoT integration system. The main objective is to assess the performance, scalability, and reliability of the proposed system with different configurations, workloads, and access strategies. The tests were designed to quantify the trade-offs between performance efficiency, latency, throughput, and concurrency.

The evaluation is divided into two complementary stages: A standardized benchmarking using Hyperledger Caliper, which provides a controlled and reproducible assessment of network behavior; A custom workload testing, designed to emulate realistic IoT data-flow conditions using sensor-based datasets.

The following sections first describe the datasets and metrics adopted, then detail the benchmarking process under the two database configurations (LevelDB and CouchDB). A comparative analysis is subsequently presented, followed by custom IoT-specific evaluations and a discussion of the obtained results in light of the objectives defined in Chapter 3.

4.1 Experimental Setup and Metrics

This Section presents an overview of the data used in the evaluation of the system, as well as how it was structured. The metrics used for evaluating the performance of the system are also depicted in the following sections.

4.1.1 Dataset and Data Model

In typical IoT environments, there are multiple sensors inside an edge layer which then communicate through various protocols the respective information. This communication is usually done through data streaming, also known as real time data processing, or through a batch approach, where chunks of data are periodically sent. From the literature, we know that batch processing is usually more efficient for processing large volumes of data, with the trade-off of a higher latency. Data streaming systems are usually more complex and use frameworks like *Kafka* or *MQTT*, which is a very popular and lightweight protocol used for IoT. It works by having publishers that produce data, in this case sensors, topics which can be defined as streams of organized data for subscribers, which in our case would be the blockchain peers or another module, to consume from.

For this work, we did not have available data at our disposal for batch or streaming. In the pursuit of understanding how real world data would fit to the proposed system, we relied on a widely used dataset from the public repository *Kaggle*. Entitled *IoT Temperature Data* [Chu18], this dataset is composed of **97600 entries** from real world IoT Sensors, monitoring room temperature and adjacent variables.

This is a simple, but practical dataset that allows us to model a real world scenario to a proposed data model. It's divided into five attributes:

- **id** - unique identifier for each reading;
- **room_id/id** - the room id in which device was installed (inside and/or outside);
- **noted_date** - date and time of reading;
- **temp** - temperature readings;
- **out/in** - whether reading was taken from device installed inside or outside of room

This dataset has been thoroughly used for forecasting time-series models regarding temperature evolution. For our specific use case these are the reasons we found it to be relevant:

1. **Real World IoT Environments** - mirrors real world environments and readings;
2. **Representative Data Volume** - the dataset contains close to 100.000 records, allowing for a comprehensive benchmarking over various realistic conditions.
3. **Availability and reproduction** - this dataset is widely used and publicly open, meaning that each experiment can be reproduced and improved on or validated.

Data Model

Following the architecture described in Chapter 3, the dataset was adapted into data structure which can be easily replicated in the ledger, where each sensor reading was represented as a digital asset stored on the ledger. The conversion process transformed each dataset record into a JSON object as shown in listing 3.1.

To optimize querying efficiency and maintain a temporal ordering of data, a composite key schema was implemented in the form of sensorID:timestamp, ensuring unique identification of each reading and facilitating range queries across specific time intervals.

Two main data modeling strategies were analyzed:

Flat Model : In this model, each sensor reading is treated as an independent blockchain asset. Every entry from the dataset corresponds to a single transaction that writes a new key-value pair to the ledger. The key is generated as a composite identifier in the form `sensorID:timestamp`, ensuring that each record remains unique and chronologically ordered.

Advantages:

- Simplified key-value mapping with direct access through primary keys.
- Predictable and uniform read/write performance across the dataset.
- Efficient compatibility with both LevelDB and CouchDB state databases.

Disadvantages:

- Increased number of ledger entries, resulting in higher world state size.
- Potentially larger disk consumption due to high transaction frequency.

Append Model : In the append-based approach, each sensor corresponds to a single blockchain asset, and new readings are appended to a list within the existing record. This method aims to reduce the number of individual ledger entries by aggregating readings per device.

Advantages:

- Compact data representation for devices generating frequent sequential readings.
- Reduced number of transactions recorded on the ledger.

Disadvantages:

- Progressive performance degradation as the internal list grows in size.
- Inefficient for long-running devices due to JSON serialization overhead.
- Limited query granularity, as individual readings cannot be indexed or queried directly.

The Flat Model was ultimately adopted for performance evaluation due to its superior scalability, predictable access patterns, and compatibility with both LevelDB and CouchDB backends. This approach also aligns with the state-transition model of Hyperledger Fabric, where each transaction represents an immutable update to the system's state.

4.1.2 Evaluation Metrics

To evaluate the performance of the proposed system in a consistent and reproducible way, we adopted the performance indicators defined by the *Hyperledger Performance and Scale Working Group (PSWG)* [**?**]. These standardized metrics provide a common baseline for reporting blockchain performance and ensure that the results obtained in this dissertation can be compared with other similar works.

The selected metrics capture both the efficiency and the reliability of the network, focusing on how quickly and accurately transactions are processed and how the system behaves under different workloads.

Transaction Throughput (TPS)

Transaction throughput represents the rate at which valid transactions are successfully committed to the blockchain. It is expressed in transactions per second (TPS) and indicates the overall capacity of the system to process requests. High throughput reflects a scalable system with an efficient resource utilization, particularly relevant in IoT environments where large volumes of data may be continuously generated and devices have limited resources.

Transaction Latency

Transaction latency measures the total time elapsed between the moment a client submits a transaction and the moment it is confirmed and visible across the network. It registers the added delay introduced by endorsement, ordering, validation, and block propagation. Low latency implies a more responsive and time-efficient system, an essential characteristic for real-time IoT scenarios.

Failure Rate

The failure rate reflects the number of rejected or invalid transactions due to endorsement policy violations, syntax errors, or version mismatches. Monitoring this allows for understanding how the network behaves under load and whether specific configurations introduce instability.

Together, these metrics provide a comprehensive view of the system's performance and operational behaviour. They allow to accurately measure scalability, responsiveness and reliability of the system. These metrics form the foundation for the benchmarking phase, where we test multiple configurations in order to understand what the best fit for the different use cases.

4.2 Benchmarking Strategy

To evaluate the performance of the developed system, a series of controlled experiments were conducted using Hyperledger Caliper. Caliper is the official benchmarking framework of the Hyperledger ecosystem, designed to provide a standardized and repeatable way to measure blockchain performance. It allows predefined workloads to be executed against a running network, recording metrics such as throughput, latency, and transaction success rate under controlled test conditions.

4.2.1 Benchmarking Environment

All benchmarking experiments were executed in the same controlled environment used for system implementation. The network was deployed through Docker Compose using the official test-network configuration provided by Hyperledger Fabric v2.5.9. The setup included one ordering service implementing the RAFT consensus protocol and two peer organizations, each hosting a single peer node.

Caliper was deployed as a separate container and connected to the Fabric gateway through the gRPC interface. The tool generated client workloads, collected response times, and aggregated the resulting statistics. This setup emulates a real-world scenario in which multiple external clients concurrently interact with the blockchain network through a gateway API.

The detailed information on the system can be found in section 3.4.1. In regards to the configurations used on Caliper, we used between 5 and 20 concurrent processes (workers), emulating client operations, and from 10 to 100 TPS, to better understand network stress. All configurations were tested using LevelDB and CouchDB.

Workload Definition

Caliper benchmarks are defined through configuration files that describe both the blockchain network configuration and the sequence of operations to be executed. Each configuration specifies the chaincode functions to invoke, the rate and duration of transactions, and the number of workers.

For this dissertation, the *Marbles* sample application, included in the Caliper repository, was adopted as the baseline workload. This benchmark models a simple lifecycle of asset creation, transfer, and history retrieval, which can be directly mapped to IoT sensor interactions. The following operations were evaluated:

- **Asset creation (initMarble)**: simulates the registration of new IoT sensor readings.
- **Asset reading (readMarble)**: represents the retrieval of individual sensor values.
- **Asset transfer (transferMarble)**: models updates or state changes in existing data.
- **Historical query (getHistoryForMarble)**: retrieves all state transitions for a given asset.
- **Rich query (queryMarblesByOwner)**: evaluated only in the CouchDB configuration, representing multi-attribute queries.

In addition to the existing transaction rate, Caliper allows us to define run times, from where the transactions would have to be sent. In this work we used values between 20 and 60 seconds.

Caliper present reports of the recorded metrics for each benchmark, which were used to construct the comparative tables that define the results for this work.

This work focuses on two main configurations of the ledger's state database to better understand the impact it has on multiple scenarios, from less demanding to a fully stressed network. In the next sections we will go over the results for LevelDB and CouchDB under the same environments.

It is most relevant to mention that the benchmarks can always be limited by the conditions of the environment, hardware and software, where they were ran.

4.2.2 Benchmarking Hyperledger Fabric with LevelDB Using Caliper

To assess the baseline performance of Hyperledger Fabric under a key-value storage configuration, we conducted a series of benchmark tests using LevelDB, the default state database. The evaluation utilized the official *marbles* chaincode provided by Hyperledger Caliper. This allowed us to isolate Fabric's native performance characteristics in a controlled, repeatable environment before introducing rich-query capabilities via CouchDB.

Baseline Benchmark Configuration

The Caliper benchmark was structured into multiple transaction rounds, each targeting a specific access or modification pattern:

- **init**: Asset creation using `initMarble`, with 500 marbles inserted at a fixed rate of 25 TPS.
- **read-by-key**: Lookup operations using `readMarble`, simulating point queries for IoT devices.
- **getHistory**: Historical reads using `getHistoryForMarble`, retrieving state transitions for a given marble.
- **transfer**: State updates using `transferMarble`, representing value mutation over time.

All tests were executed with 5 concurrent workers under a fixed-rate strategy. Results were measured in terms of throughput (TPS), latency, Max and Average, and transaction failure rate.

Results Summary (LevelDB)

Round	TPS	Avg Latency	Max Latency	Failures	Notes
`init`	25.2	0.15 s	0.27 s	0	Smooth high throughput asset creation
`read-by-key`	10.5	0.01 s	0.02 s	0	Very fast key lookups
`getHistory`	5.3	0.01 s	0.02 s	0	Efficient for low history depth
`transfer`	9.3	0.34 s	2.05 s	0	Higher latency from state mutation

Table 4.1: Performance results for LevelDB-based Caliper benchmark.

The results confirm that LevelDB offers excellent performance for direct key access and high throughput asset creation. Historical access using `getHistoryForMarble` was also performant, although initially limited in insight due to low state depth per asset. Transfer operations introduced slightly higher latency, which is expected due to endorsement and commit delays, yet remained stable without failures.

Effect of Historical Depth on `getHistoryForMarble` Query

To analyze how increased state depth affects historical queries, we re-ran the `getHistoryForMarble` benchmark after executing a round of `transferMarble` operations. These updates caused each marble asset to maintain multiple state transitions in the ledger. As shown in Table 4.2, the average query latency increased from 0.01 to 0.02 seconds, and the maximum latency tripled from 0.02 to 0.06 seconds. However, throughput remained constant at 5.3 TPS, and no transaction failures were recorded. This suggests that LevelDB handles deeper history queries with only a modest increase in access time, maintaining excellent reliability and performance under small-to-moderate ledger depths.

Run Context	Avg Latency (s)	Max Latency (s)	Throughput (TPS)	Failures
Before Transfers	0.01	0.02	5.3	0
After Transfers	0.02	0.06	5.3	0

Table 4.2: Comparison of `getHistoryForMarble` performance before and after state depth increased.

Notably, no performance degradation in throughput or transaction success rate was observed, underscoring LevelDB's stability even under increased query complexity.

Scalability of Asset Creation at Scale (10,000 entries)

To evaluate how Hyperledger Fabric with LevelDB handles large-scale asset registration, we extended the `initMarble` benchmark to 10,000 transactions. The test was executed with 5 concurrent workers at a fixed throughput of 25 TPS. Table 4.3 shows that all 10,000 transactions succeeded, without failures, and average latency remained at 0.15 seconds, identical to the baseline test with only 500 assets. This confirms that asset creation via simple key based writes, scales linearly in Fabric under the tested configuration.

Metric	Value
Successful Transactions	10,000
Failures	0
Avg Latency (s)	0.15
Max Latency (s)	0.29
Throughput (TPS)	25.0

Table 4.3: Performance results for 10,000 asset insertions at 25 TPS.

Scaling Transaction Rate During Transfer Stress

To evaluate how the Hyperledger Fabric network responds to increasing transaction throughput, we performed a series of transfer stress tests at 10, 25, 50, and 100 TPS using 5 concurrent workers. Each test maintained a fixed duration of 20 seconds. As shown in Table 4.4, throughput scaled nearly linearly with the configured TPS, and no transaction failures were recorded across any test.

Interestingly, average latency decreased as TPS increased, improving from 0.35 seconds at 10 TPS to just 0.07 seconds at 100 TPS. This suggests that higher submission rates help streamline client-server communication and reduce idle time between requests. However, maximum latency remained consistent across all runs, hovering around 2.04 seconds, possibly reflecting peak endorsement or commit duration under occasional queueing pressure. It might also be an environment limitation.

TPS Target	Success	Avg Latency (s)	Max Latency (s)	Throughput (TPS)	Failures
10	205	0.35	2.05	9.3	0
25	505	0.17	2.04	22.9	0
50	1005	0.11	2.05	45.6	0
100	2005	0.07	2.04	91.0	0

Table 4.4: Transfer performance under increasing transaction rates (5 workers).

Client Concurrency and Throughput Stability

To evaluate the impact of increased client concurrency, we re-ran the 10,000 transaction asset creation benchmark with 20 workers instead of 5. As shown in Table 4.5, the network maintained identical throughput (25 TPS) and zero failures in both cases. However, average latency improved from 0.15 to 0.09 seconds, suggesting that higher worker counts help distribute client load more efficiently across Fabric peers. Slight increases in maximum latency (from 0.29 to 0.33 seconds) were observed. These findings suggest that Fabric scales horizontally with increased clients without compromising stability or throughput.

Workers	TPS	Avg Latency (s)	Max Latency (s)	Failures
5	25.0	0.15	0.29	0
20	25.0	0.09	0.33	0

Table 4.5: Comparison of `initMarble` benchmark performance under varying client concurrency.

High Frequency State Updates: Transfer Stress Test

To simulate a real-time IoT scenario with frequent state changes, we executed a sustained update benchmark using the `transferMarble` function at 100 transactions per second over 60 seconds. With 20 concurrent workers and 6020 successful transactions, the network maintained near perfect throughput (100.1 TPS) and experienced no transaction failures. Average latency remained at 0.07 seconds, while maximum latency peaked at 0.84 seconds. These results confirm that Hyperledger Fabric can reliably handle high-throughput mutation workloads, making it suitable for continuous sensor data ingestion or status propagation at scale.

Metric	Value
Successful Transactions	6020
Failures	0
Throughput (TPS)	100.1
Avg Latency (s)	0.07
Max Latency (s)	0.84

Table 4.6: Performance metrics for transfer stress test (100 TPS over 60s).

Concurrency Effects on Historical State Queries

To assess the concurrency impact on historical queries, we increased the number of clients querying `getHistoryForMarble` from 5 to 20, while keeping the transaction duration and TPS fixed (15 seconds at 5 TPS). As shown in Table 4.7, the average latency increased from 0.01 to 0.04 seconds, and maximum latency rose to 0.22 seconds. However, throughput improved slightly and no transaction failures occurred. While concurrency introduces some delay in historical state retrieval, the system remains performant and stable under moderate parallel access.

Notably, the total number of transactions executed increased from 80 to 100. This is not due to an increase in the global TPS target, which remained fixed at 5. Rather, the rise is explained by Caliper's internal scheduling: when more workers are active, the transaction dispatching mechanism becomes more responsive and efficient, often resulting in a slightly higher number of submissions within the fixed time window. This behavior reflects improved concurrency handling and is consistent with Caliper's asynchronous workload model.

Workers	Throughput (TPS)	Avg Latency (s)	Max Latency (s)	Failures
5	5.3	0.01	0.02	0
20	6.2	0.04	0.22	0

Table 4.7: Impact of worker concurrency on `getHistoryForMarble` query performance.

These results serve as a reference point for later comparison against CouchDB, where rich query capabilities are introduced at the cost of performance overhead.

4.2.3 Benchmarking Hyperledger Fabric with CouchDB Using Caliper

To assess the performance implications of enabling rich-query capabilities in Hyperledger Fabric, we conducted a series of benchmark tests using CouchDB as the state database. CouchDB supports JSON document storage and allows expressive querying using selectors. While this introduces added flexibility, it also increases the computational and I/O overhead during ledger operations.

The tests mirrored those run in the LevelDB phase to enable fair comparison. All rounds were executed with 5 concurrent workers using a fixed-rate strategy.

Results Summary (CouchDB)

Round	TPS	Avg Latency	Max Latency	Failures	Notes
`init`	25.2	0.18 s	0.29 s	0	Slightly higher latency than LevelDB; stable throughput
`read-by-key`	10.4	0.12 s	0.14 s	0	Key lookups show modest increase in latency
`getHistory`	5.3	0.01 s	0.02 s	0	Historical reads unaffected by state DB type
`transfer`	9.3	0.36 s	2.07 s	0	Consistent latency profile during mutation
`query-rich`	5.3	0.02 s	0.06 s	0	Rich query performed successfully; slightly higher latency

Table 4.8: Performance results for CouchDB-based Caliper benchmark.

The results show that CouchDB supports baseline operations effectively, with only modest increases in latency when compared to LevelDB. The average latency for asset creation rose from 0.15s to 0.18s, and key lookups exhibited a minor increase from 0.01s to 0.12s. Rich query execution using JSON selectors was successful after correcting the input format, highlighting the need for precise query string handling. Historical queries and transfer operations remained stable, indicating that CouchDB's additional indexing complexity does not significantly affect these interactions at low concurrency levels.

Scalability of Asset Creation at Scale (10,000 entries, CouchDB)

To evaluate how Hyperledger Fabric with CouchDB handles large-scale asset registration, we extended the `initMarble` benchmark to 10,000 transactions. The test was executed with 5 concurrent workers at a fixed throughput of 25 TPS.

Round	TPS	Avg Latency	Max Latency	Failures	Notes
`init-marble-10k`	25.0	0.18 s	0.40 s	0	CouchDB handles scaled insert workload without congestion

Table 4.9: Performance results for CouchDB-based 10,000 asset insertions at 25 TPS.

The results indicate that CouchDB sustains its write performance even as the ledger grows, showing no transaction failures and maintaining throughput identical to the 500 entry baseline. The average latency remained stable at 0.18 seconds, which is identical to the smaller scale CouchDB test, confirming that asset creation via simple key based writes continues to scale linearly even under document-based state storage.

Client Load and Throughput Scaling with CouchDB (10,000 entries)

To analyze the network's responsiveness under increasing transaction rates, we benchmarked the `transferMarble` operation using 10,000 assets and varied the TPS between 10 and 100. Each test was executed with 5 concurrent workers over a fixed duration of 20 seconds. As seen in Table 4.10, CouchDB maintained linear throughput scaling with no transaction failures. Latency values decreased with higher TPS, indicating effective utilization of available compute and I/O capacity.

Round	TPS	Avg Latency	Max Latency	Failures	Notes
`transfer-10tps`	9.3	0.37 s	2.08 s	0	Stable mutation under minimal load
`transfer-25tps`	22.9	0.19 s	2.09 s	0	Nearly linear throughput; latency halves
`transfer-50tps`	45.5	0.13 s	2.07 s	0	Efficient scaling; max latency stable
`transfer-100tps`	90.7	0.10 s	2.11 s	0	High-throughput execution; excellent parallelism

Table 4.10: Performance results for CouchDB-based transferMarble benchmark with 10,000 assets.

Client Concurrency and Throughput Stability (CouchDB)

To evaluate the effect of increased client concurrency on asset creation, we re-ran the `initMarble` benchmark with 20 workers instead of 5. As shown in Table 4.11, the network sustained the target throughput of 25 TPS with zero transaction failures. Interestingly, average latency decreased from 0.18 seconds (observed at 5 workers) to 0.12 seconds, while maximum latency remained moderate. These results suggest that CouchDB scales horizontally under load, distributing client traffic efficiently and improving responsiveness under parallel insert workloads.

Workers	TPS	Avg Latency (s)	Max Latency (s)	Failures	Notes
5	25.0	0.18	0.29	0	Baseline CouchDB insert performance
20	25.0	0.12	0.47	0	Improved latency from concurrent load balancing

Table 4.11: Comparison of initMarble performance under varying client concurrency with CouchDB.

High-Frequency State Updates: Transfer Stress Test (CouchDB)

To simulate a real time IoT scenario with continuous state changes, we executed a sustained benchmark using the `transferMarble` function at 100 transactions per second over 60 seconds. The test was performed with 20 concurrent workers. As shown in Table 4.12, the system successfully processed all 6020 transactions with zero failures. Throughput closely matched the target rate, while average latency remained modest at 0.21 seconds.

Metric	Value
Successful Transactions	6020
Failures	0
Throughput (TPS)	100.1
Avg Latency (s)	0.21
Max Latency (s)	1.97

Table 4.12: Performance metrics for CouchDB-based transfer stress test (100 TPS over 60s).

The results confirm that Hyperledger Fabric with CouchDB remains performant and stable even under high mutation throughput. While latency increased compared to LevelDB (from 0.07s to 0.21s average), the network achieved consistent delivery without degradation or failures, validating its viability for continuous data ingestion workloads.

Concurrency Effects on Historical State Queries (CouchDB)

To evaluate how concurrent access impacts historical state retrieval, we increased the number of clients invoking the `getHistoryForMarble` function from 5 to 20. The TPS and duration were held constant (5 TPS for 15 seconds). As shown in Table 4.13, average latency increased from 0.01 to 0.04 seconds, and maximum latency rose to 0.18 seconds. Throughput improved slightly. The absence of transaction failures confirms that CouchDB remains robust under moderate read concurrency.

Workers	Throughput (TPS)	Avg Latency (s)	Max Latency (s)	Failures	Notes
5	5.3	0.01	0.02	0	Baseline historical query latency
20	6.2	0.04	0.18	0	Slight latency increase under load; stable performance

Table 4.13: Impact of worker concurrency on getHistoryForMarble query performance (CouchDB).

Scalability of Rich Queries Under Concurrency (CouchDB)

To examine CouchDB's performance with indexed attribute based queries under concurrent load, we benchmarked the `queryMarblesByOwner` function using 20 workers at a fixed rate of 10 transactions per second. The results, shown in Table 4.14, indicate stable throughput and low latency, with no transaction failures. These findings suggest that CouchDB handles indexed rich queries efficiently even under moderate concurrency.

Workers	TPS	Avg Latency (s)	Max Latency (s)	Failures	Notes
20	10.2	0.02	0.29	0	Rich query by indexed field executed reliably under parallel load

Table 4.14: Performance of CouchDB `queryMarblesByOwner` under concurrency (20 workers).

4.2.4 Comparison: LevelDB vs. CouchDB Performance

The benchmarking results obtained from both configurations revealed clear and consistent performance differences between LevelDB and CouchDB, confirming the trade-offs expected between execution speed and query flexibility in Hyperledger Fabric.

Throughput and Latency

LevelDB consistently achieved lower latency and higher throughput across all test scenarios. In asset creation and transfer operations, average latency remained around 0.15 seconds, while CouchDB showed slightly higher delays of approximately 0.18 to 0.21 seconds under identical workloads. Since LevelDB is embeded into the peer node, we believe the difference arises primarily from the additional HTTP communication between Fabric peers and the external CouchDB service, as well as the JSON serialization overhead required for document storage.

Throughput values mirrored this pattern: in LevelDB, the network sustained nearly linear scalability from 10 to 100 transactions per second, maintaining a success rate of 100%. CouchDB exhibited similar stability but with marginally reduced throughput, around 90–95% of the LevelDB rate, particularly at higher concurrency levels. Nonetheless, this reduction is modest and falls within acceptable operational margins for most IoT-oriented applications.

Concurrency Behaviour

As client concurrency increased, both configurations demonstrated robust scalability. In LevelDB, higher worker counts reduced average latency due to improved parallel request handling and reduced idle time between transactions. In CouchDB, a comparable trend was observed, although with slightly more variance most likely because it is an external database. Importantly, no transaction failures or system instability occurred in either configuration, highlighting the maturity and resilience of Hyperledger Fabric's RAFT ordering mechanism.

Historical Queries

When analysing historical queries `getHistoryForMarble`, both LevelDB and CouchDB achieved nearly identical performance, with average latencies between 0.01 and 0.04 seconds even as concurrency increased. This implies that historical reads rely primarily on Fabric's internal ledger for historical records, which is always a LevelDB support database. This suggest that the possible bottleneck will most likely be on the internal LevelDB database, and not the adjacent state database.

Rich Queries

CouchDB provided a significant advantage in rich queries. Using JSON selectors and secondary indexes, it allowed the execution of attribute searches `queryMarblesByOwner`, which are not supported in LevelDB. These queries performed well under moderate concurrency, with latencies averaging 0.02 seconds and no failed transactions. While slightly slower than simple key lookups, this functionality introduces powerful analytical capabilities, particularly valuable for IoT systems that may require filtering data by sensor type, location, or timestamp range.

4.3 Overall Evaluation

This section presents the evaluation of the proposed blockchain–IoT system, focusing on how the implemented data model, smart-contract logic, and state database selection contribute to achieving the architectural goals outlined in Chapter 3. Whereas Section 4.2 provided a baseline performance analysis through standard benchmarking scenarios, the following evaluation aims to validate the proposed system design under realistic IoT data interactions. The intent is not only to measure throughput and latency, but to demonstrate how the underlying data structures and database configuration support traceability, flexibility, and scalability.

4.3.1 Experimental Dataset and Data Model

The dataset employed in this evaluation derives directly from the structure previously defined in Section 4.1.1. It represents a simplified yet representative sample of IoT sensor data, which was used to synthetically generate data for caliper, to emulate temperature readings collected across multiple rooms within a monitored facility. Each record follows the same schema defined earlier, containing the fields `sensorID`, `roomID`, `timestamp`, `temperature`, and `location`.

This structure used was the "append" logic, maintained in a JSON format to mirror realistic IoT data payloads and to align with the document-oriented capabilities of CouchDB. Each document corresponds to a sensor or measurement event and is uniquely identified by a composite key (`sensorID:timestamp`). Such a model allows efficient chronological ordering and facilitates range-based queries, which are essential for time-series analysis in IoT systems.

A key design decision in this work was to persist most operational data directly within the *world state*, enabling immediate access to the most recent sensor readings without reconstructing them from the ledger history. This approach offers significant performance advantages for real-time applications, as Fabric's state database provides indexed retrieval and query functions that dramatically reduce latency. From a security standpoint, however, it was necessary to ensure that this accessibility does not compromise the integrity of the ledger. Since the world state is a transient reflection of the committed blockchain, all updates remain cryptographically anchored in the immutable ledger, ensuring auditability. In addition, Hyperledger Fabric's endorsement and access control mechanisms guarantee that only authorized peers and clients can modify or query the data, maintaining the confidentiality and consistency of the stored information.

This design therefore provides an interesting balance between efficiency and security. By leveraging CouchDB's document-based representation, the system gains flexible query capabilities while still benefiting from Fabric's deterministic validation and endorsement processes. The result is a data model that supports high-frequency IoT updates and temporal analytics while maintaining the integrity guarantees inherent to permissioned blockchains.

4.3.2 Implementation and Interaction Logic

Building upon the dataset described in Section 4.3.1, the functional logic of the proposed system was implemented through a custom Hyperledger Fabric chaincode. The objective was to establish a modular and transparent framework capable of handling IoT data ingestion, storage, and retrieval while ensuring traceability and deterministic execution. The chaincode was developed in GO and deployed across two organizational peers, each maintaining a synchronized world state through CouchDB.

The smart contract encapsulates five primary functions representing the complete lifecycle of sensor information within the blockchain network:

- `CreateSensor`: Registers a new sensor in the ledger, storing its unique identifier, room assignment, and installation metadata. This establishes the base entity that subsequent readings can reference.
- `ReadSensor`: Retrieves the latest version of a sensor's metadata directly from the world state, providing fast access to its current configuration or status.
- `UpdateSensor`: Alters existing sensor metadata, such as location or configuration parameters. This operation affects only the sensor's descriptive attributes, not its recorded measurements, which remain immutable once created.
- `GetHistory`: Uses Fabric's internal key-history database to retrieve the full evolution of a given sensor's metadata. This mechanism provides complete provenance for configuration changes or maintenance events.
- `GetIntervalReadingsCQ`: Executes range-based queries through CouchDB's Mango selectors to fetch all measurement records associated with a sensor within a specified time interval. Each reading is an independent asset, identified by a composite key (`sensorID:timestamp`), and therefore remains unalterable after being committed to the ledger.

This functional design establishes a clear separation between sensor-level metadata and reading-level events. Metadata may evolve through controlled updates, while individual readings are immutable, ensuring that no measurement data can be tampered with or replaced. The endorsement policy required validation from both organizational peers for every transaction, ensuring consistency and preventing unauthorized state modifications.

By leveraging CouchDB as the state database, the system gains native support for JSON selectors and composite-key indexing, enabling efficient temporal and analytical queries. Each transaction is reflected immediately in the world state for fast access, yet remains cryptographically anchored in the ledger, maintaining the immutability and auditability required for trust in IoT environments. This dual-layered approach captures both operational efficiency and data integrity, aligning the system with blockchain's foundational principles of traceability and distributed trust.

4.3.3 Evaluation of the Data Model

This subsection evaluates the behaviour of the proposed data model and CouchDB state database under full dataset execution. The experiments were conducted on the dataset defined in Section 4.1.1, comprising approximately 100 000 individual measurement records distributed across multiple sensors and rooms. Each reading was represented as an independent asset identified by a composite key (`sensorID:timestamp`), resulting in a world state of roughly the same scale. This configuration aimed to emulate a realistic IoT deployment in which large volumes of sensor data are continuously ingested and queried in near real time.

Experimental goal. The purpose of this benchmark was to validate that the document-based data model and the chosen state database can sustain high insertion volumes and maintain predictable latency as the world state grows. The test also assessed the query responsiveness of the network under realistic temporal and analytical operations. This evaluation complements the generic *Marbles* benchmark discussed in Section 4.2, extending the analysis to the specific case of IoT data management and traceability.

Workload design. Five representative operations were benchmarked using Hyperledger Caliper with five concurrent workers and fixed-rate controllers: `CreateSensor`, `ReadSensor`, `UpdateSensor`, `GetHistory`, and `GetIntervalReadingsCQ`. The `CreateSensor` and `UpdateSensor` functions operated on the full dataset, while `GetHistory` and `GetIntervalReadingsCQ` were used to query subsets of the 100 000 readings based on temporal conditions. The CouchDB state database was configured with indexed fields for `sensorID` and `timestamp` to optimise range-based queries and key lookups.

Table 4.15: Performance metrics for the IoT-Sensor benchmark with ∼100k world-state records (CouchDB, 5 workers).

Operation	TPS	Avg. Latency	Max Latency	Failures	Notes
CreateSensor	23.8	0.18 s	0.41 s	0	Bulk sensor registration over 100k entries; latency increased slightly due to ledger commit but remained stable.
ReadSensor	9.8	0.01 s	0.04 s	0	Constant-time key lookups
UpdateSensor	9.3	0.20 s	0.84 s	0	Controlled metadata updates on 100k sensors; moderate endorsement cost, no MVCC conflicts.
GetHistory	192.4	0.01 s	0.05 s	0	Provenance queries executed efficiently through Fabric's key-history DB; unaffected by dataset size.
GetIntervalReadingsCQ	9.5	0.03 s	0.09 s	0	Temporal range queries across full dataset; indexed fields ensured predictable latency.

Interpretation. The results demonstrate that the system maintained stable throughput and low latency even when operating over a world state exceeding 100 000 documents. Insertions through `CreateSensor` and metadata modifications via `UpdateSensor` showed slight latency growth compared to smaller tests, primarily due to ledger commit operations, but remained within acceptable margins for near real-time IoT processing.

Query-oriented functions retained near-constant performance. `ReadSensor` delivered sub-50 ms access times for direct key lookups, confirming that CouchDB's primary-key indexing is resilient to state growth. `GetIntervalReadingsCQ` exhibited marginal latency increases relative to dataset volume but remained efficient, with range queries completing in under 100 ms per operation. Notably, `GetHistory` performance remained largely unaffected by dataset size, as its execution depends only on the number of historical states associated with a given key, not the total ledger volume.

Implications for the data model. These findings reinforce the design hypothesis that storing IoT readings as immutable, JSON-based documents within the world state provides both flexibility and efficiency. The combination of CouchDB's document model with Hyperledger Fabric's endorsement and commit pipeline allows continuous ingestion at scale without compromising traceability. The slight increase in write latency at large state sizes is compensated by the system's ability to support analytical and temporal queries directly on-chain, removing the need for off-chain data replication. Overall, the evaluation confirms that the chosen data representation and state database configuration are well-suited for high-volume IoT environments, maintaining both performance stability and ledger-level security guarantees.

4.3.4 Access Methods: Client vs. Chaincode

When comparing access methods, direct chaincode invocation presented lower latency because it bypassed the additional processing steps required by the REST gateway. In contrast, client-side requests through the Fabric Gateway introduced measurable overhead due to JSON serialization and HTTP communication. Nonetheless, this overhead remained acceptable for practical deployments, where gateway abstraction provides an essential layer of modularity and security between IoT devices and the blockchain network.

4.3.5 Summary

The evaluation conducted in this section validated the effectiveness of the proposed blockchain–IoT architecture and its underlying data model under realistic conditions. By employing the dataset previously defined in Section 4.1.1, consisting of approximately 100 000 sensor readings, the system was tested at a scale representative of real-world IoT deployments. The results confirmed that the document-oriented data model, implemented through CouchDB, and the modular chaincode logic jointly supported reliable ingestion, querying, and historical tracking of sensor data.

Across all benchmarked operations, throughput and latency remained stable and predictable, demonstrating that the world state can accommodate substantial data growth without degrading system responsiveness. The combination of Fabric's deterministic consensus, endorsement policies, and key-history database ensured that each operation was both verifiable and immutable. Furthermore, the use of CouchDB enabled efficient range queries and analytical operations directly within the blockchain state, validating one of the core objectives of this work, enabling flexible and transparent access to IoT data while preserving ledger-level integrity.

From a design perspective, separating immutable sensor readings from mutable sensor metadata proved to be a critical architectural decision. This structure allowed the network to support continuous data inflow while maintaining the immutability guarantees required for trustworthy historical reconstruction. The results therefore confirm that the proposed data model and state database configuration achieve an effective balance between scalability, data integrity, and analytical expressiveness.

Overall, the system evaluation demonstrates that the developed architecture performs reliably under realistic data volumes, providing a foundation for secure and auditable IoT data management within a permissioned blockchain environment. The following section discusses these results in depth, interpreting their broader implications for scalability, design trade-offs, and the feasibility of blockchain integration in IoT ecosystems.

4.4 Discussion

The experimental results demonstrate that Hyperledger Fabric provides a robust and flexible foundation for managing IoT data, offering both performance scalability and operational reliability under varying workloads. The benchmarking and IoT oriented evaluations reveal important trade-offs between performance efficiency and query flexibility, directly influenced by the choice of state database and data model. While we can't completely rule out the impact of the environment where the benchmarks and test took place, they are fully comparable, and expected to proportionally replicate throughout other systems.

Performance Behaviour

Across all Caliper benchmarks, the system achieved stable throughput and low latency, even under increasing transaction rates and concurrency. The RAFT consensus mechanism ensured deterministic transaction ordering, and no data loss or failures occurred throughout the experiments.

LevelDB exhibited the highest throughput and lowest latency due to its embedded architecture and efficient key–value storage. CouchDB, while marginally slower, maintained consistent performance with only minor overheads introduced by JSON document processing and external communication through its REST interface. This overhead was expected, given that CouchDB operates as an independent process rather than embedded as with LevelDB.

The near linear scalability observed in both configurations confirms that the proposed architecture can handle moderate, or high, IoT data ingestion rates. This is particularly relevant for environments where thousands of devices generate frequent updates, as the system maintained operational stability without exceeding peer or orderer resource limits.

Impact of Data Modeling

The experiments with different data structures confirmed the importance of selecting a schema aligned with the expected IoT workload. The *flat key model*, where each reading is stored as an individual state entry identified by a composite key (`sensorID:timestamp`), proved superior to the append-list model. The flat approach offered predictable access times and avoided payload size limitations inherent to the gRPC communication layer.

The append-list model, although conceptually appealing for continuous data streams, resulted in larger transactions and higher serialization costs, making it unsuitable for real time IoT ingestion. These findings highlight the relevance of maintaining high data granularity in distributed ledgers where each update constitutes an independent state transition.

Read and Query Performance

Read operations, both in the benchmarking and in the custom IoT evaluations, were consistently faster than write operations. This behaviour aligns with the expected dynamics of Hyperledger Fabric, where read requests do not trigger consensus or state validation. Historical queries exhibited almost identical latency in both LevelDB and CouchDB, demonstrating that such operations depend primarily on Fabric's internal historical database, rather than the world-state database.

CouchDB's advantage emerged in rich queries, which enabled attribute filtering and complex JSON selectors. Although these queries incurred slightly higher processing time, they provided meaningful capabilities for IoT analytics and monitoring use cases, such as retrieving all readings from a specific location or time window.

System Overhead and Gateway Interaction

The gateway abstraction introduced an expected, yet limited, increase in latency when compared to direct chaincode invocation. This was primarily due to additional serialization and HTTP communication overhead. However, the practicality of this test was to better understand how much the abstraction, and exposition, of the Fabric Gateway would impact the operations.

The measurements confirmed that this additional overhead remained within acceptable bounds, ensuring that system remained responsiveness.

Relevance to IoT Integration

Taken together, these results validate the suitability of permissioned blockchain frameworks such as Hyperledger Fabric for IoT data management. The network demonstrated consistent scalability, integrity, and determinism, critical properties for distributed sensor systems that must ensure data immutability and traceability.

The observed trade-off between *LevelDB's performance* and *CouchDB's flexibility* provides an important design guideline: while the former should be prioritised for high frequency, latency sensitive scenarios, the latter is more advantageous when query diversity or analytical capabilities are required.

These findings also confirm that Fabric's modular architecture can effectively accommodate IoT specific requirements through adjustable components such as the state database, chaincode logic, and gateway interface.

Summary

The discussion presented in this Chapter consolidates the experimental and analytical findings of this work, linking quantitative performance results with the qualitative architectural choices that guided the system's design. Together, these results validate the initial research premise that a permissioned blockchain, when coupled with an appropriate data model and a document-oriented state database, can provide a viable and efficient foundation for IoT data management.

The experiments demonstrated that the Hyperledger Fabric network, configured with CouchDB as its state database, sustains stable throughput and latency across different workloads and dataset sizes. The baseline *Marbles* benchmark confirmed the expected performance characteristics of Fabric under simple asset-transfer logic, while the custom *IoT-Sensor* benchmark extended these tests to more realistic, data-intensive workloads. Despite the larger dataset and more complex operations, the system maintained consistent transaction rates and predictable response times, confirming that the selected architecture scales effectively for high-volume sensor data.

Beyond quantitative measures, the evaluation also revealed important architectural insights. The decision to distinguish immutable sensor readings from mutable sensor metadata proved essential for balancing traceability with operational flexibility. This design prevents concurrency conflicts, reduces state contention, and ensures that all measurements remain permanently anchored to the ledger, fulfilling the immutability and provenance requirements central to blockchain-based IoT systems. Furthermore, CouchDB's integration with Fabric enabled direct execution of temporal and range-based queries, demonstrating that analytical capabilities can be achieved within the blockchain layer itself.

Overall, the combined results indicate that the proposed architecture achieves a functional equilibrium between decentralization, integrity, and performance. The system operates reliably under realistic IoT data conditions, supports transparent and auditable data handling, and maintains the security properties expected of enterprise-grade distributed ledgers. These findings not only validate the architectural design and data model proposed in this work but also provide empirical evidence that permissioned blockchain frameworks can feasibly support IoT ecosystems requiring both operational efficiency and verifiable data provenance.

5 Conclusion and Future Work

This dissertation presented the design, implementation, and evaluation of a blockchain based architecture for Internet of Things (IoT) decentralization using Hyperledger Fabric. The work aimed to analyse the feasibility of employing distributed ledger technologies to ensure integrity, traceability, and scalability in environments where data are generated continuously by heterogeneous devices.

Throughout the development process, a complete permissioned blockchain network was implemented, including the configuration of peer and ordering nodes, the development of tailored chaincode, and the integration of a REST-based gateway that simulated the interaction between edge devices and the blockchain infrastructure. The system was deployed in a controlled environment, enabling the execution of multiple benchmarking and custom testing scenarios.

The experimental component was divided into two major phases. The first comprised standardised benchmarking using the Hyperledger Caliper framework, allowing the quantitative measurement of performance metrics such as throughput, latency, and transaction success rate. This stage provided a controlled baseline to compare the performance of different system configurations and to assess the behaviour of the network under increasing workloads and concurrency levels.

The second phase focused on the evaluation of the proposed system implementation, using actual sensor datasets and custom workloads that reproduced the lifecycle of sensor readings in distributed environments. This approach made it possible to analyse how Hyperledger Fabric behaves when handling batches of data, range queries, and historical reconstructions typical of IoT applications.

The results obtained demonstrated that the proposed system is capable of maintaining stable and predictable performance under diverse operating conditions. Both LevelDB and CouchDB configurations achieved consistent throughput and 100% transaction success, although with distinct behavioural patterns.

LevelDB delivered superior latency and transaction rates, confirming its efficiency for high frequency ingestion workloads, while CouchDB introduced a moderate overhead but enabled rich and attribute queries through its JSON document model. These findings validate that the choice of state database has a direct and measurable impact on the balance between raw performance and data retrieval flexibility.

The analysis of the data modelling approaches further confirmed that a flat key structure, defined as `sensorID:timestamp`, offers the most scalable and robust representation for IoT readings. This model avoided payload limitations, reduced serialization costs, and simplified query execution, outperforming the append-list alternative when handling large datasets. The results also revealed that historical queries depend mainly on Fabric's internal history database, rather than the world state database, maintaining stable latency across configurations.

The gateway abstraction, which mediated external access to the blockchain network, introduced a small but consistent overhead compared to direct chaincode invocations. However, this architectural layer ensured modularity, interoperability, and potential integration with other IoT or data analytics systems, thus reinforcing the design's practicality for real world deployment.

Overall, the experimental results validate the viability of Hyperledger Fabric as a technological backbone for IoT data integrity and distribution. The network maintained determinism, immutability, and high reliability across all test scenarios, while remaining flexible enough to adapt to different query and storage models. By combining the controlled benchmarking phase with realistic IoT data evaluations, this work demonstrated how blockchain technology can be effectively tuned to meet the operational demands of IoT ecosystems.

Future Work

Although the results were encouraging, several aspects remain open for future exploration. Firstly, all experiments were conducted within a controlled local environment; extending the deployment to a geographically distributed network would provide a more accurate representation of latency and communication overhead. Secondly, only the RAFT consensus mechanism was evaluated; alternative protocols, such as BFT-SMaRt, could be analysed to assess trade-offs between fault tolerance, throughput, and energy efficiency. Additionally, the integration of real time data ingestion through streaming protocols such as MQTT or Kafka would allow for a more faithful representation of continuous IoT data flows. Future iterations of this work could also explore hybrid architectures combining on-chain verification with off-chain data storage mechanisms, such as IPFS or conventional databases, to improve scalability while preserving data provenance, whilst empowering the system with an analytical capacity. Finally, privacy and compliance aspects, particularly concerning GDPR, should be addressed through the study of selective data disclosure, encryption, and access control mechanisms to ensure regulatory conformity in large scale IoT deployments.

In conclusion, the research conducted in this dissertation demonstrates that permissioned blockchain systems, and particularly Hyperledger Fabric, can meet the fundamental requirements of IoT data management. When properly configured and modelled, these systems are capable of delivering both operational efficiency and the trust guarantees that decentralised environments demand. The outcomes achieved lay the groundwork for future developments in blockchain-enabled IoT infrastructures, paving the way for secure, transparent, and scalable data ecosystems.

Bibliography

[ABB$^+$18] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolic, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: A distributed operating system for permissioned blockchains. *CoRR*, abs/1801.10228, 2018.

[ACAH22] Zachary Auhl, Naveen Chilamkurti, Rabei Alhadad, and Will Heyne. A comparative study of consensus mechanisms in blockchain for iot networks. *Electronics*, 11(17), 2022.

[AMQ13] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. Rbft: Redundant byzantine fault tolerance. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pages 297–306, 2013.

[BMZ18] L. M. 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)*, pages 1545–1550, 2018.

[BRS$^+$25] B Bhasker, P Muralidhara Rao, P Saraswathi, S Gopal Krishna Patro, Javed Khan Bhutto, Saiful Islam, Mohammed Kareemullah, and Addisu Frinjo Emma. Blockchain framework with iot device using federated learning for sustainable healthcare systems. *Scientific Reports*, 15(1), July 2025.

[CAD18] Aleksey Charapko, Ailidani Ailijiang, and Murat Demirbas. Bridging paxos and blockchain consensus. In *iThings/GreenCom/CPSCom/SmartData* [CAD18], pages 1545–1552.

[Chu18] Nelson Chu. Iot temperature data. `https://www.kaggle.com/datasets/nelsonchu/iot-temperature-data`, 2018. Accessed: 2025-09-09.

[CL02] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, November 2002.

[Cra21] Karl Crary. Verifying the hashgraph consensus algorithm. *CoRR*, abs/2102.01167, 2021.

[CWW$^+$18] Zhuan Cheng, Gang Wu, Hao Wu, Muxing Zhao, Liang Zhao, and Qingfeng Cai. Deterministic proof of work. *CoRR*, abs/1808.04142, 2018.

[DBCB$^+$22] Tomás Domínguez-Bolaño, Omar Campos, Valentín Barral, Carlos J. Escudero, and José A. García-Naya. An overview of iot architectures, technologies, and existing open-source projects. *Internet of Things*, 20:100626, October 2022.

[ECMB14] A. Elmangoush, H. Coskun, T. Magedanz, and N. Blum. The openmtc platform: An open source m2m/iot service platform compliant with etsi m2m and onem2m standards. *IEEE Communications Magazine*, 52(12):94–101, 2014.

[EIP18] Nabil El Ioini and Claus Pahl. *A Review of Distributed Ledger Technologies: Confederated International Conferences: CoopIS, CTC, and ODBASE 2018, Valletta, Malta, October 22-26, 2018, Proceedings, Part II*, pages 277–288. 10 2018.

[fab]

[GGD$^+$25] Mozhgan Gholami, Ali Ghaffari, Nahideh Derakhshanfard, Nadir iBRAHIMOĞLU, and Ali Asghar. Blockchain integration in iot: Applications, opportunities, and challenges, Apr 2025.

[GKR18] Peter Gazi, Aggelos Kiayias, and Alexander Russell. Stake-bleeding attacks on proof-of-stake blockchains. In *CVCBT* [GKR18], pages 85–92.

[GNLF$^+$22] Joao Giao, Artem A. Nazarenko, Fernando Luis-Ferreira, Diogo Gonçalves, and Joao Sarraipa. A framework for service-oriented architecture (soa)-based iot application development. *Processes*, 10(9):1782, September 2022.

[grp] Grpc.

[Gur21] Rajesh Gurbani. Iot: Ongoing challenges and opportunities in mobile technology. *International Journal of Electrical, Electronics and Computers*, 6:1–9, 01 2021.

[HAA$^+$24] Ehtisham Ul Haque, Waseem Abbasi, Ahmad Almogren, Jaeyoung Choi, Ayman Altameem, Ateeq Ur Rehman, and Habib Hamam. Performance enhancement in blockchain based iot data sharing using lightweight consensus algorithm, Nov 2024.

[HMZ20] Dongyan Huang, Xiaoli Ma, and Shengli Zhang. Performance analysis of the raft consensus algorithm for private blockchains. *IEEE Trans. Syst. Man Cybern. Syst.*, 50(1):172–181, 2020.

[HS91] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *J. Cryptol.*, 3(2):99–111, 1991.

[IDB18] Johannes Innerbichler and Violeta Damjanovic-Behrendt. Federated byzantine agreement to ensure trustworthiness of digital manufacturing platforms. In *CRYBLOCK@MobiSys* [IDB18], pages 111–116.

[IEE20] Ieee standard for an architectural framework for the internet of things (iot). *IEEE Std 2413-2019*, pages 1–269, 2020.

[Int12] International Telecommunication Union. Overview of the internet of things. ITU-T Recommendation Y.2060, June 2012. Available at https://www.itu.int/rec/T-REC-Y.2060-201206-I/en.

[JUM$^+$16] Guth Jasmin, Breitenbucher Uwe, Falkenthal Michael, Leymann Frank, and Reinfurt Lukas. Comparison of iot platform architectures: A field study based on a reference architecture. *IEEE Conference Proceedings*, 2016:6, January 2016.

[KAMA22] Moez Krichen, Meryem Ammi, Alaeddine Mihoub, and Mutiq Almutiq. Blockchain for modern applications: A survey. *Sensors*, 22:5274, 07 2022.

[KMM$^+$18] Seoung Kyun Kim, Zane Ma, Siddharth Murali, Joshua Mason, Andrew Miller, and Michael Bailey. Measuring ethereum network peers. In *Internet Measurement Conference* [KMM$^+$18], pages 91–104.

[KPBC22] Yavuz Selim Kıyak, Alex Poor, Isil Budakoglu, and Ozlem Coskun. Holochain: a novel technology without scalability bottlenecks of blockchain for secure data exchange in health professions education. *Discover Education*, 1, 09 2022.

[KTZ19] Sachin Kumar, Prayag Tiwari, and Mikhail Zymbler. Internet of things is a revolutionary approach for future technology enhancement: a review. *Journal of Big Data*, 6, 12 2019.

[Lia20] Ying-Chang Liang. *Blockchain for Dynamic Spectrum Management*, pages 121–146. 01 2020.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[Mig18] Sara Migliorini. Enhancing blockchain smart-contracts with proof-of-location. 2018.

[MMA23] Raouf Mehannaoui, Kinza Nadia Mouss, and Karima Aksa. Iot-based food traceability system: Architecture, technologies, applications, and future trends. *Food Control*, 145:109409, 2023.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[one24] oneM2M Partnership Project. onem2m; functional architecture. Technical Report TS-0001 Version 5.3.0, oneM2M, 2024. Available at https://www.onem2m.org/technical/published-specifications.

[OO14] Diego Ongaro and John K. Ousterhout. In search of an understandable consensus algorithm. In Garth Gibson and Nickolai Zeldovich, editors, *USENIX Annual Technical Conference*, pages 305–319. USENIX Association, 2014.

[PG] Hyperledger Performance and Scale Working Group. Hyperledger blockchain performance metrics white paper: Hyperledger.

[PVM24] Andrea Petrucci, Maël Vial, and Vincent Magnin. IoT Big data processing: what approaches to address the challenge? In *Proceedings of the NTDE24 Workshop*, Lausanne, Switzerland, 2024.

[QCM$^+$22] Hongwu Qin, Yuntao Cheng, Xiuqin Ma, Fei Li, and Jemal Abawajy. Weighted byzantine fault tolerance consensus algorithm for enhancing consortium blockchain efficiency and security. *Journal of King Saud University - Computer and Information Sciences*, 34(10, Part A):8370–8379, 2022.

[RYKK21] Ziaur Rahman, Xun Yi, Ibrahim Khalil, and Andrei Kelarev. Blockchain for iot: A critical analysis concerning performance and scalability, Nov 2021.

[SCP$^+$20] Amritraj Singh, Kelly Click, Reza M. Parizi, Qi Zhang, Ali Dehghantanha, and Kim-Kwang Raymond Choo. Sidechain technologies in blockchain networks: An examination and state-of-the-art review. *Journal of Network and Computer Applications*, 149:102471, 2020.

[SDL$^+$25] Najmus Sakib Sizan, Diganta Dey, Abu Layek, Ashraf Uddin, and Eui-Nam Huh, Feb 2025.

[SP22] Bela Shrimali and Hiren B. Patel. Blockchain state-of-the-art: architecture, use cases, consensus, challenges and opportunities. *Journal of King Saud University - Computer and Information Sciences*, 34(9):6793–6807, 2022.

[SZA$^+$22] Jie Song, Pengyi Zhang, Mohammed Alkubati, Yubin Bao, and Ge Yu. Research advances on blockchain-as-a-service: architectures, applications and challenges. *Digital Communications and Networks*, 8(4):466–475, 2022.

[SZJS22] Reza Soltani, Marzia Zaman, Rohit Joshi, and Srinivas Sampalli. Distributed ledger technologies and their applications: A review. *Applied Sciences*, 12(15), 2022.

[SZL$^+$24] Weihu Song, Mengxiao Zhu, Dong Lu, Chen Zhu, Jiejie Zhao, Yi Sun, Lei Li, and Haogang Zhu. Blockchain bottleneck analysis based on performance metrics causality, Oct 2024.

[TdOCF$^+$19] Marcela Tuler de Oliveira, Gabriel Carrara, Natalia Fernandes, Célio Albuquerque, Ricardo Carrano, Dianne Medeiros, and Diogo Menezes. Towards a performance evaluation of private blockchain frameworks using a realistic workload. pages 180–187, 02 2019.

[TKE22] Amal Tawakuli, Daniel Kaiser, and Thomas Engel. Transforming iot data preprocessing: A holistic, normalized and distributed approach. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems (SenSys '22)*, Boston, MA, USA, November 2022.

[YMA22] J. Yusoff, Z. Mohamad, and M. Anuar. A review: Consensus algorithms on blockchain. *Journal of Computer and Communication*, (3):37–50, 2022.

[ZCDV17] Jun Zhou, Zhenfu Cao, Xiaolei Dong, and Athanasios V. Vasilakos. Security and privacy for cloud-based iot: Challenges. *IEEE Communications Magazine*, 55(1):26–33, 2017.