UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Iwada Eja Bassey

Blockchain in Edge - Cloud Computing Continuum

Master's Thesis (30 ECTS)

Supervisor(s): Chinmaya Kumar Dehury, PhD
Mubashar Iqbal, PhD

Tartu 2023

Blockchain in Edge - Cloud Computing Continuum

Abstract:
In the era of Edge Computing, Internet of Things(IoT) devices connect and communicate to create a network of objects that can collect information from the environment without human intervention. While the IoT offers many benefits, it also poses several cybersecurity risks, including the lack of detection of malicious IoT sensors, weak authorization, and authentication protocols, and insecure management of data received from IoT devices. Developing new solutions to enhance IoT sensors' security to address these issues is crucial. One crucial challenge that requires attention is the secure management and storage of data collected from IoT devices at the Edge. Many existing solutions rely on centralized systems vulnerable to tampering and must provide reliable data traceability records. To address this challenge, this thesis proposes a blockchain-based architecture for securing and managing data collected from IoT devices. By leveraging blockchain technology, we create a distributed data storage architecture that eliminates the need for centralized network topologies. This approach offers several advantages, including immutability, decentralization, distributivity, enhanced security, transparency, instant traceability, and increased efficiency through automation. Our results demonstrate that this proposed architecture provides a high level of performance and can be used as a scalable, massive data storage solution for IoT devices using blockchain technologies. One significant advantage of this approach is that new IoT sensors can quickly be enrolled and unenrolled in our architecture without retrofitting. Our system does not depend on any specific communication protocol and can be applied to any IoT application. In summary, our proposed architecture provides a robust and secure solution for managing and storing data collected from IoT devices, thus enhancing the overall security of the IoT devices in the Edge.

Keywords:
Blockchain, Edge computing, Fog computing, Data Privacy, Hyperledger Fabric, Cloud Computing

CERCS:
P170 - Computer Science, Numerical Analysis, Systems, Control

Plokiahel Edge'is - Pilvandmetöötluse kontinuum

Lühikokkuvõte:

Edge Computingu ajastul, esemevõrgu(IoT) seadmed ühenduvad ja suhtlevad, et luua objektide võrgustik, mis suudab koguda informatsiooni keskkonnast inimeste sekkumiseta. Kuigi IoT omab palju eeliseid, kannab see ka endas mitmeid küberturvalisuse riske, sealhulgas pahatahtlike IoT andurite tuvastamise puudumine, nõrk autoriseerimine, autentimise protokollid ja ebaturvaline IoT seadmetelt vastu võetud andmete haldamine. Nende probleemide lahendamiseks on ülioluline uute lahenduste väljatöötamine IoT andurite turvalisuse suurendamiseks. Üks oluline väljakutse, mis nõuab tähelepanu, on IoT seadmetest Edge-is vastu võetud andmete turvaline haldamine ja hoiustamine. Paljud olemasolevad lahendused põhinevad tsentraliseeritud süsteemidel, mis on haavatavad rikkumiste suhtes ja peavad pakkuma usaldusväärseid andmete jälgitavuse kirjeid. Selle väljakutse lahendamiseks pakub see lõputöö välja plokiahelapõhise arhitektuuri IoT-seadmetest kogutud andmete turvamiseks ja haldamiseks. Plokiahela tehnoloogiat võimendades loome hajutatud andmesalvestusarhitektuuri, mis välistab vajaduse tsentraliseeritud võrgutopoloogiate järele. Sellel lähenemisviisil on mitmeid eeliseid, sealhulgas muutumatus, detsentraliseerimine, jaotus, suurem turvalisus, läbipaistvus, kohene jälgitavus ja automatiseerimise kaudu suurem tõhusus. Meie tulemused näitavad, et see kavandatud arhitektuur tagab kõrge jõudluse ja seda saab kasutada plokiahela tehnoloogiaid kasutavate IoT seadmete skaleeritava massilise andmesalvestuslahendusena. Selle lähenemisviisi üheks oluliseks eeliseks on see, et uusi IoT andureid saab kiiresti meie arhitektuuris registreerida ja registrist eemaldada ilma moderniseerimiseta. Meie süsteem ei sõltu ühestki konkreetsest sideprotokollist ja seda saab rakendada igale IoT rakendusele. Kokkuvõtteks võib öelda, et meie pakutud arhitektuur pakub tugevat ja turvalist lahendust IoT seadmetest kogutud andmete haldamiseks ja salvestamiseks, suurendades seeläbi Edge'i IoT seadmete üldist turvalisust.

Võtmesõnad:

Plokiahel, Edge Computing, Udu Andmetöötlus, Andmete Privaatsus, Hyperledger Fabric, Pilvtehnoloogia

CERCS:

P170 – arvutiteadus, numbriline analüüs, süsteemid, juhtimine

Contents

List of Tables

List of Figures

List of Code Snippets

List of Abbreviations

HLF	Hyperledger Fabric
PBN	Primary Blockchain Network
SBN	Sensor Blockchain Network
E2C-Block	Blockchain for Edge to Cloud Continuum
dApp	decentralized applications
API	Application Programming Interface
SDK	Software Development Kit
TPS	Transaction Per Second
IoT	Internet of Things
JSON	JavaScript Object Notation
XML	Extensible Markup Language
HTTPS	Hypertext Transfer Protocol Secure
RDBMS	Relational Database Management System
AWS	Amazon Web Services
CA	Certificate Authority

1 Introduction

Blockchain, Fog Computing, and Edge Computing technologies have emerged as significant technological innovations in recent years [23]. These technologies offer unique benefits and can be combined to create new solutions and applications previously impossible. With blockchain technology, users can conduct secure and transparent transactions without intermediaries. In contrast, edge computing allows for real-time processing and data analysis closer to where it is generated [18].

However, integrating these technologies presents several challenges that must be addressed to realize their full potential. One such challenge is scalability, as the decentralized nature of blockchain networks can result in slower transaction speeds and higher costs than traditional centralized systems [27]. Moreover, the increasing amount of data generated from edge devices and the complexity of processing and analyzing it require innovative approaches. Additionally, security is a crucial concern when dealing with sensitive data, particularly in the case of edge computing, where devices are often deployed in unsecured environments [27].

One challenge that needs addressing is the issue of trust in edge computing environments. Since edge devices are often distributed and unsecured, there is a need to establish trust between devices, nodes, and systems [23]. Another challenge is the efficient management of data generated by edge devices. The large volume of data these devices generate requires efficient storage, processing, and management techniques [1]. Additionally, the heterogeneity of edge devices presents challenges in terms of interoperability and data standardization [23].

Researchers have proposed various solutions to address these challenges, such as integrating blockchain technology with edge and cloud computing [27]. The use of blockchain can enhance security and trust in edge computing environments by providing a decentralized and tamper-proof ledger for recording transactions and storing data [27]. While most of these solutions address the secure data transfer from the edge to the blockchain network, what happens if these IoT devices' data get manipulated in transit? Considering the scalability issues inherent with blockchain, do we store all this extensive data from these edge devices on the Blockchain network? How efficient would it be to issue read queries to this blockchain network?

This thesis proposes E2C-Block - an architecture incorporating blockchain with an external data repository to efficiently collect, securely store, and process IoT data from various IoT sensors. This architecture leverages blockchain's security and immutability features to ensure the data's integrity and privacy. Additionally, it employs edge and cloud computing to provide efficient data processing and management capabilities, thus addressing the challenges of data heterogeneity, trust, and efficient data management in the edge-cloud continuum.

1.1 Research Problem

Integrating blockchain, fog computing, and edge computing presents several challenges that must be addressed to realize their full potential. One significant challenge is scalability, which results in slower transaction speeds and higher costs than traditional centralized systems. Moreover, the increasing amount of data generated from edge devices and the complexity of processing and analyzing it require innovative approaches. The heterogeneity of edge devices presents challenges regarding interoperability and data standardization. Security is another crucial concern when dealing with sensitive data, particularly in the case of edge computing, where devices are often deployed in unsecured environments. Trust is a significant challenge in edge computing environments, where trust is needed between devices, nodes, and systems. Efficient management of data generated by edge devices is another challenge, as the large volume of data requires efficient storage, processing, and management techniques. This thesis aims at answering these research questions:

- What are the best security measures for edge computing environments to protect sensitive data?
- How can blockchain, fog computing, and edge computing be integrated to overcome scalability and reduce storage costs?
- How can blockchain technology be leveraged to ensure the integrity and immutability of data stored in fog and edge computing environments?

1.2 Research Method

The research in this thesis employed a design science approach, beginning with a comprehensive review of the existing literature on blockchain technology, fog computing, and edge computing to identify the research problem and objectives. A conceptual framework was developed to guide the research process, which included defining the scope of the study and outlining the research questions that need to be addressed.

We then designed the proposed architecture, considering the integration of blockchain technology, MinIO cloud-based storage system, and sensor networks for efficient data management and security. The design process involved several iterations to ensure the architecture's effectiveness and practicality.

Next, the architecture was implemented and tested using a prototype system with two Blockchain networks. We also integrated an external data repository to provide a reliable off-chain storage solution that complements blockchain networks.

Finally, we conducted a series of tests and simulations to measure the proposed architecture's performance and scalability. The tests focused on assessing the architecture's ability to ensure data integrity, prevent tampering, and reduce storage costs. The results of these tests and simulations were analyzed to identify the proposed architecture's strengths and weaknesses and identify areas for improvement.

Large Language Models, particularly ChatGPT, were utilized during the research process to obtain quick summaries of various research papers. Additionally, Grammarly was extensively used to correct minor grammar errors in the thesis.

1.3 Contribution

The proposed E2C-Block architecture contributes towards ensuring the security and integrity of IoT sensor data by leveraging blockchain technology and MinIO cloud-based storage system. The use of blockchain technology provides a secure and tamper-proof system for processing, transmitting, and storing sensor data. The Sensor Blockchain network adds an extra layer of security by authenticating the data before being sent to the Primary Blockchain network. Moreover, keeping the hash of the sensor data on the Primary Blockchain network instead of the data itself reduces the storage footprint significantly while ensuring data integrity. Section 3 introduces and covers these two blockchain networks and the Minio Storage Server in great detail.

Furthermore, the integration of MinIO provides an efficient and reliable off-chain storage solution that complements blockchain networks. It enables large amounts of sensor data to be stored reliably and accessed when needed. By utilizing this approach, organizations can maintain the integrity of their data and build a trusted system that enhances their overall security posture. In cases where it is necessary to confirm the authenticity of a specific data point, the data can be retrieved from the MinIO server and compared against the hash value previously stored on the Primary Blockchain network. This process ensures that the data has not been tampered with or altered.

The architecture takes a comprehensive approach to data security, guaranteeing that the data is processed, transmitted, and stored securely and tamper-proof. This approach provides a dependable and effective method of protecting confidential information and preventing unauthorized access, ultimately enhancing the security and privacy of the system.

1.4 Thesis Outline

Section 1 introduces the thesis and presents basic concepts. We define the research questions this thesis addresses and our research methodology. Additionally, a brief discussion on the potential contributions of the thesis is included.

Section 2 introduces Edge, Fog and Cloud Computing, and Blockchain technologies and discusses the components and types of blockchain networks. We compare Corda and Hyperledger Fabric (HLF) and explain why we chose HLF as our reference implementation.

In Section 3, we extensively delve into the design of our proposed architecture. Our discussion entails exploring the options we considered, alternative possibilities, and the reasoning behind our ultimate design choices.

Section 4 provides the implementation of the design for the proposed architecture. It covers the various technical aspects of the implementation process, such as the programming languages, frameworks, tools used to build the architecture, and any challenges encountered during the implementation and how they were overcome.

Section 5, discusses the experimental setup and describes the various experiments conducted to evaluate our proposed architecture. We also present our experimental setup, define some key metrics and list all the experiments done.

Section 6 presents and discusses the findings, highlighting significant observations and trends observed during the experiments in Section 5. Additionally, this section provides a critical evaluation of the results, including a discussion of the implications of the findings and their relevance to the proposed architecture's performance.

Section 7 explains how the work carried out in this thesis provides solutions to the research questions that we initially identified and stated. It also explores potential future research directions for this thesis, as well as the limitations of the study.

Section 8, serves as the thesis's conclusion, summarizing the findings and their implications for the proposed architecture. This section provides a final assessment of the effectiveness of the proposed architecture in addressing the research problems based on the experimental results discussed in Section 6.

2 Background

This section discusses edge, fog, and cloud computing and provides an overview of blockchain technology, specifically HLF. We conclude with a comparison between HLF and Corda.

2.1 Edge, Fog, and Cloud Computing

Edge computing is a distributed computing paradigm that brings processing and data storage closer to the point of demand to speed up response times and conserve bandwidth. This approach involves deploying computer resources like servers and storage devices near data sources or users who need access at the network's edge. Minimizing latency and the time between requesting and receiving data is a key advantage of edge computing [7]. To improve the processing, analysis, and response to data, edge computing moves computer resources closer to the edge, which is critical for real-time processing applications like autonomous vehicles or automated industrial systems. By shifting some workloads from centralized resources like cloud servers to processing data closer to the edge, edge computing can reduce network congestion and enhance overall performance while lowering cloud computing costs [5]. Another benefit of edge computing is its capacity to manage significant volumes of data generated by IoT devices like sensors and smart appliances, which require real-time processing and analysis. Deploying computing capabilities at the network's edge can improve data management efficiency, reduce latency, and save bandwidth [23].

However, edge computing poses various challenges, for example, ensuring data security and privacy in a distributed system and managing and sustaining distributed computing resources. As edge computing involves positioning resources, careful design and implementation are essential for optimum performance and efficiency. Fog computing is a distributed architecture that extends cloud computing to the network's edge. It allocates computing, storage, and networking resources between cloud data centers and edge-connected hardware, including routers, gateways, and IoT devices. This will improve performance and efficiency as data processing is done closer to where the data is being produced. At the same time, latency can be reduced, and network capacity can be preserved [4].

Edge computing and fog computing are similar concepts but have distinct implementations and use cases [15]. Edge computing involves placing resources at the network's edge, closer to data sources or users needing access. In contrast, fog computing is a layer of computing that sits between edge devices, such as sensors and smart appliances, and the cloud. It processes and stores data generated by these devices, making it ideal for IoT applications where a large amount of data is generated by devices distributed over a wide area [15, 4]. While edge computing and fog computing aim to bring computing power closer to the data source, they have different use cases. Fog computing is deployed when a hybrid cloud and edge architecture is required and when more processing power than an edge device can provide is necessary for data processing. On the other hand, edge computing is commonly used when data processing can be completed entirely on a local device without cloud services [15].

Cloud computing is a popular computing model that delivers computing resources over the Internet, including servers, storage, and software [2]. It has become a critical part of modern IT infrastructure due to its scalability, accessibility, and cost savings. One significant benefit of cloud computing is its ability to scale computing resources up or down as needed without incurring additional costs associated with owning and managing computer resources [2]. This enables organizations to respond quickly to changing business requirements, which can be especially valuable for businesses with fluctuating demand or seasonal spikes in demand. Cloud computing also allows companies to reduce their capital expenditures associated with owning and managing hardware and infrastructure [2], making it an attractive option for businesses.

Fog computing is a complementary technology to cloud computing that provides additional resources and processing power for applications that require real-time processing and analysis of large volumes of data. It is particularly useful for IoT applications requiring low latency and real-time decision-making capabilities. IoT devices can process and analyze data locally, reducing data transmission to cloud data centers. This can enhance data privacy and security, reduce network congestion and latency, and improve overall application performance [4]. Both edge computing and fog computing aim to reduce latency and improve application performance, but they have different use cases and implementations compared to cloud computing.

Figure 1 describes the hierarchical relationship between edge, fog, and cloud computing environments. The image shows that these computing environments are interconnected, with cloud computing at the top of the hierarchy, fog computing at the bottom edge computing at the bottom. It also clearly represents how these computing environments can work together to provide a comprehensive computing infrastructure that meets the needs of various applications and use cases.

2.2 Blockchain

Blockchain technology has become increasingly popular in recent years due to its ability to secure and verify data in a decentralized manner. A blockchain is a distributed database that allows multiple parties to record transactions in a secure, verifiable, and permanent way. Each block contains a timestamp and a link to the previous block, forming a chain. This technology was first introduced in 2008 by Satoshi Nakamoto in the white paper "Bitcoin: A Peer-to-Peer Electronic Cash System" [19].

One of the key features of blockchain technology is that it is decentralized, meaning that a single entity or organization does not control it. Instead, it relies on a network of computers known as nodes to validate and record transactions. This makes it more

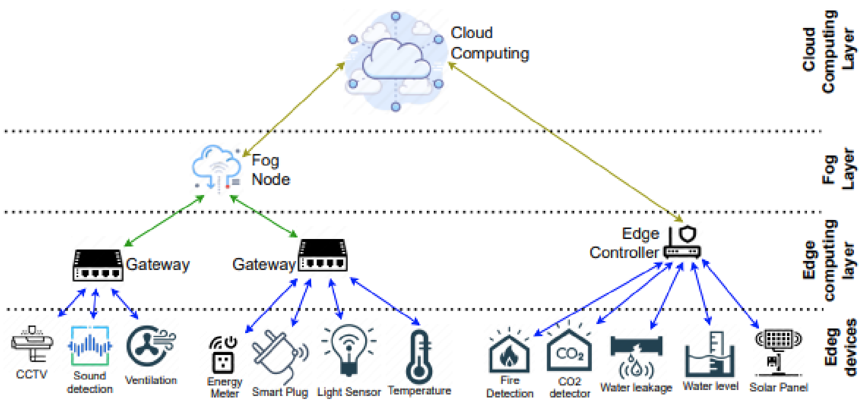


Figure 1. Representation of edge, fog, and cloud computing environments in a hierarchical manner, adapted from [6]

.

secure and transparent, as it is challenging to alter or tamper with the data on the chain. Moreover, blockchain technology provides a new level of trust and transparency to the digital world. It allows for a distributed consensus where every online transaction, past and present, involving digital assets can be verified at any time [27].

One area where blockchain technology has the potential to revolutionize the way we secure and manage data is the Internet of Things (IoT). IoT devices have become increasingly popular in recent years [3], and blockchain technology can play an essential role in securing the data generated by these devices. By using blockchain to store and verify IoT data, we can ensure the authenticity and integrity of the data, as well as prevent unauthorized access or manipulation. This is particularly important in industries such as healthcare and finance, where the security and privacy of data is critical.

The healthcare industry, in particular, can benefit significantly from blockchain technology. A blockchain system can help manage patient data and ensure the privacy and security of medical records [20]. It can also give patients more control over their health data, such as deciding who can access it and for what purpose. Blockchain technology can also help to prevent medical identity theft, a growing problem in the healthcare industry.

In addition, blockchain technology can be used in various industries, such as logistics, supply chain management, and finance. For instance, Walmart uses blockchain to track the origin of products like vegetables and fruits, ensuring their quality and safety [13]. HLF is a permissioned blockchain platform for enterprise use cases that provides a modular architecture enabling the development of smart contracts and decentralized applications (dApps) with high flexibility and confidentiality [15]. HLF is one of the most widely used platforms for developing enterprise blockchain applications. It has been adopted by major companies in various industries, including finance, healthcare, and supply chain management.

One key emerging use case of blockchain technology involves "smart contracts." Smart contracts are computer programs that can automatically execute the terms of a contract. They can also include conditional statements, allowing for the automatic execution of specific actions based on fulfilling certain conditions. In HLF, smart contracts are implemented using a variety of programming languages, such as Golang, Javascript, and Java. These smart contracts are packed into what HLF refers to as chain codes and deployed on the blockchain network.

2.3 Types of Blockchains

There are three main types of blockchains: private, public, and permissioned [27]:

Private blockchains: Private blockchains are restricted to a specific group of participants. Access to the network and the data stored on it is controlled by a single entity or organization, meaning only approved participants can join the network and access the data. Organizations often use private blockchains to share data and facilitate collaboration among trusted parties securely [27]. One example of a private blockchain is Corda, developed by R3. Corda [21] is designed for multiple parties in a specific business network, such as a consortium of banks. The network administrator controls access to the network and the stored data, ensuring that only approved participants can join the network and access the data.

Public blockchains: Public blockchains are open to anyone to participate in and access. For example, anyone can join the network, access blockchain data, and validate and record transactions. Public blockchains are decentralized and rely on distributed nodes to validate and record transactions (e.g., Bitcoin and Ethereum). Bitcoin is one of the most well-known examples of a public blockchain. Anyone can participate in the Bitcoin network by downloading the software and contributing computing power. Transactions are validated and recorded by nodes on the network, and the data is publicly accessible to anyone with an internet connection. [27]

Permissioned blockchains: Permissioned blockchain is a type of blockchain that sits somewhere between private and public blockchains. While anyone can join a permissioned blockchain, access to the network and its stored data is restricted to approved participants. Only authorized participants can validate and record transactions, although anyone can still access the data stored on the blockchain [27]. Permissioned blockchains are often used by organizations to securely share data and facilitate collaboration among trusted parties while still maintaining some level of control over who can access and modify the data. HLF is an example of a permissioned blockchain designed for enterprise use cases. While anyone can participate in the network, access to the data is restricted to approved participants. HLF allows organizations to create private channels within the network, which can be used to securely share data and facilitate collaboration among trusted parties while maintaining confidentiality for sensitive information. Here, I present a high-level comparison between these blockchains (Table 1).

Table 1. Comparison of Private, Public, and Permissioned Blockchains

Characteristic	**Private**	**Public**	**Permissioned**
Access	Limited	Open	Restricted
Membership	Closed	Open	Closed or Open
Consensus	Centralized	Decentralized	Centralized or Decentralized
Transaction Speed	Fast	Slow	Fast
Transaction Cost	Low	High	Low
Security	High	High	High
Immutability	High	High	High
Flexibility	Low	High	High
Governance	Controlled	Uncontrolled	Controlled
Examples	HLF, Quorum	Bitcoin, Ethereum	Corda, Ripple

2.4 Blockchain Solutions

This subsection examines several blockchain technologies, particularly HLF and Corda. We will compare these two technologies and justify selecting HLF as the reference implementation for our proposed architecture.

2.4.1 Hyperledger Fabric

HLF is an open-source blockchain platform developed by the Linux Foundation to support the development and deployment of distributed ledger applications. It is designed to be modular, scalable, and secure, making it suitable for various use cases, including supply chain management, financial services, and healthcare. It has been shown to achieve an end-to-end throughput of more than 2980 transactions per second and scale well to over 100 peers [11].

One of the key features of HLF is its modular architecture, which allows developers to plug in different components, such as consensus algorithms, membership services, and data stores, to create a customized blockchain solution. This modularity makes it possible to tailor HLF to the specific needs of different industries and applications.

Another vital feature of HLF is its support for smart contracts, self-executing contracts with the terms of the agreement between two or more parties being directly written into lines of code [9]. Smart contracts allow for the automation of specific processes and enforcing certain conditions. They can be used to facilitate, verify, and execute the negotiation or performance of a contract. In HLF, smart contracts are implemented using a variety of programming languages and deployed as chain codes on the network.

HLF also provides APIs and SDKs (Software Development Kits) that allow developers to interact with the blockchain and deploy smart contracts. These APIs and SDKs are available in various programming languages, including JavaScript, Python, and Java, making it easier for developers to build applications on top of it. HLF's architecture comprises several components that provide a robust and secure blockchain network. These components include the membership service provider, ordering service, peer nodes, and smart contracts (packaged as chain codes). These components and their roles in HLF will be discussed in Section 2.5

2.4.2 Corda

Corda is a flexible and scalable platform designed to operate with the current financial services industry and integrate with pre-existing enterprise technology. It is a permissioned ledger, asset modeling tool, and workflow routing engine. Corda enables solutions that enhance and decentralize assets while maintaining privacy and regulatory oversight [21]. Corda's primary goal is to enable businesses to create and manage contracts that can be automatically executed using smart contracts. The platform also has features for managing identity and privacy, making it ideal for financial use cases.

Unlike public blockchains, Corda is not a cryptocurrency but a platform for managing financial agreements. One of the most significant features of Corda is its focus on privacy. Corda is designed to be used within specific business networks and gives businesses control over who can access data on the network. Transactions are only visible to parties directly involved [21], making them an effective tool for managing sensitive financial information. Corda also includes features for managing identity and ensuring compliance with regulatory requirements. Businesses can create their identity and access management policies and use tools to verify and share identity data. The platform also has features for managing legal agreements and enforcing compliance with regulatory requirements. Corda is a powerful tool for managing financial agreements within business networks. Its modular architecture and privacy focus to make it a flexible and customizable platform that can be tailored to meet the needs of different industries and use cases. Additionally, its open-source nature offers a transparent and collaborative approach to building distributed ledger solutions.

2.4.3 Comparison

HLF and Corda are open-source distributed ledger technology platforms with distinct differences in design, architecture, and intended use cases. Corda primarily focuses on financial services and emphasizes privacy and control over data access, while HLF provides a modular and flexible architecture suitable for various industries.

HLF's consensus process involves nodes with different roles and tasks, including clients, peers, and endorsers, to ensure message delivery errors do not occur [9]. A pluggable algorithm is used for consensus, allowing for the use of different algorithms. On the other hand, in Corda, the consensus is achieved at the transaction level and only involves parties concerned, with notary nodes used for consensus over uniqueness [21].

Smart contracts in HLF are self-executing contracts that execute code to model contractual logic in the real world. However, its legal validity may need to be further clarified. In contrast, Corda allows smart contracts to contain legal prose, with smart legal contracts being legal prose expressed and implemented in smart contract code, providing the code with legitimacy rooted in the associated legal prose [21].HLF is a general-purpose DLT platform suitable for various use cases, while Corda is specifically designed for financial applications such as trade finance, insurance, and capital markets. Table 2 provides a quick comparison of HLF and Corda.

Table 2. Comparison of Corda 4.9 and HLF 2.4 ([21], [11])

Feature	**Corda**	**HLF**
Type of Platform	Permissioned	Permissioned
Smart Contract Language	Kotlin	Go, Java,TypeScript
Consensus Algorithm	RAFT, BFT-SMaRt	Kafka, SBFT
Privacy	High	Moderate
Scalability	High	Moderate
Transaction Throughput	200-300 tps	1000-3000 tps
Participation	Only required parties	All network nodes
Governance	R3	Linux Foundation
Hosting	Self-hosted or cloud	Self-hosted or cloud
License	Apache 2.0	Apache 2.0

2.4.4 Why Hyperledger Fabric?

In selecting the reference platform for the implementation of our proposed architectures, We considered six key points:

1. The platform should be permissioned and private.
2. It should allow for a pluggable consensus algorithm.

3. It should support smart contract implementation in a programming language we already know.

4. It should have a well-documented software development kit (SDK) for building client applications.

5. It should be interoperable with most open-source benchmarking tools.

6. It should have a high transaction per second (TPS) rate.

After considering the intended use case of the platform, we compared Fabric with Corda. We found that both met the criteria of being permissioned and private (1) and supporting pluggable consensus algorithms (2). However, HLF was preferred due to its support for smart contract development in Typescript, a language we are more comfortable with (3). While both projects lacked documentation, we found more resources on HLF, which would help implement our proposed architecture.

To validate our claims of usability, we needed a benchmarking tool. The benchmarking tools we explored were more interoperable with HLF than Corda. Additionally, white papers published by both organizations showed that HLF had a much higher Transaction Per Second (TPS) rate compared to a similar configuration with Corda. This was crucial because our proposed architecture would be used in the IoT environment where data generation rates can be incredibly high. We needed an underlying blockchain technology with a proven high TPS [11]. Finally, Corda was primarily developed for financial use cases, whereas our proposed use case needed more financial, leading us to choose HLF.

2.5 HyperLedger Fabric

This subsection comprehensively examines some fundamental components of an HLF network.

2.5.1 Peers

A vital component of an HLF Blockchain network is its peers. A Peer is a non-ordering node in an HLF network. It stores and manages copies of ledgers and smart contracts (chain codes). An Orderer is a type of node in the blockchain network that handles the interaction of applications and peers to keep a current and consistent ledger across a channel [8]. Peers are a flexible and redundant element that can be created, started, stopped, reconfigured, and deleted [8]. They also host special system chain codes which contain information about HLF Network's overall configuration. To allow client applications to connect, peers expose a set of APIs - The HLF Gateway Service - that helps these client applications interact with their services. Starting in HLF v2.4, the HLF Gateway service is installed and enabled on each Peer by default. Unlike the client application (in HLF v2.3 and earlier), the gateway service manages transaction proposals and endorsements on the Peer [8].

Peers execute a consensus protocol to validate transactions, group them into blocks, and build a chain of these blocks. In conjunction with orderers, Peers ensures the ledgers across consistent and current across a channel and, consequently, a blockchain network. In an HLF, peers are usually provided by organizations participating in the network. Figure 2 shows a high-level overview of a peer that comprises user-defined chain codes, a ledger, and system chain codes.

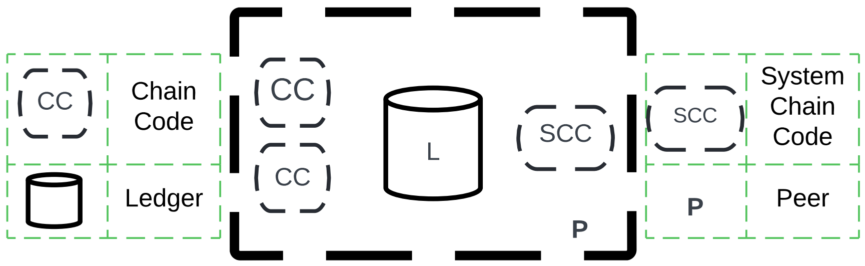


Figure 2. Overview of contents of a Peer Node.

2.5.2 Orderers

Orderer nodes are a crucial component of the HLF. They are responsible for managing the ordering service, ensuring all transactions are properly ordered and packaged into blocks. Once the blocks are created, the orderer nodes distribute them to all the participating nodes in the network. This process ensures that the ledger remains consistent and tamper-proof [10]. HLF provides three distinct implementations for its ordering service, which is designed to be modular and has a flexible consensus system that can be configured as needed.

The first implementation is the Solo ordering service, a single node that handles the ordering process for the network [10]. This implementation is useful for development and testing purposes or for small networks that do not require a high level of transaction throughput. However, it is unsuitable for more extensive networks where performance and scalability are crucial. The second implementation is the Kafka ordering service, which uses Apache Kafka to handle the ordering. Kafka allows multiple orderer nodes to be deployed in a cluster, providing higher transaction throughput and fault tolerance [10]. Kafka ordering service is suitable for more extensive networks that require high levels of performance and scalability. The third implementation is the Raft ordering service, which uses the Raft consensus algorithm to handle the ordering process. Like the Kafka ordering service, the Raft ordering service allows multiple orderer nodes to be deployed in a cluster. However, it differs in its consensus algorithm, and its architecture is more fault-tolerant [10].

The choice of ordering service implementation will depend on the network's specific performance, scalability, and fault tolerance needs. Table 3 summarises the major differences between these three implementations.

Table 3. Differences between Solo, Kafka, and Raft Ordering Services

Feature	**Solo**	**Kafka**	**Raft**
Consensus	Solo	Kafka	Raft
Scalability	Low	High	High
Fault tolerance	None	Limited	High
Ordering speed	Fast	Slow	Fast
Message ordering	N/A	Partial order	Total order
Endorsement	Optional	Mandatory	Mandatory
Network size	Small	Large	Large
Operational cost	Low	High	High

2.5.3 Ledgers

In HLF, the ledger is another vital component that records the current state of business transactions and the transactions that led to them [12]. One of the fundamental features of a blockchain ledger is that information regarding an object's past is never subject to change, even if the object's current state may change. HLF's ledger consists of two interconnected components: a world state and a blockchain. Figure 3 illustrates the composition of an HLF Ledger, which is broken down into a world state and a blockchain; the content of the blockchain determines the world state. Each component represents a set of details regarding a collection of business objects.

The world state is a database containing ledger states' most recent values [12]. Rather than requiring a complete transaction log to calculate a state's current value, the world state makes it simple for a program to access it directly. Key-value pairs are the default way ledger states are expressed, but HLF offers flexibility. The world state can often change due to the creation, modification, and deletion of states. Therefore, a database is used to implement the global state, which offers a wide range of operators for storing and retrieving states effectively.

HLF's world state database can be implemented using CouchDB, which is useful when ledger states are structured as JSON documents. The default database for the World state is LevelDB in HLF [12]. However, for the implementation of our proposed architecture, we used CouchDB. Figure 4 shows the world state containing one state with a kV pair value: *key=2023-04-02T12-Tartucitycouncil-SensorOne, value = "temperature":*

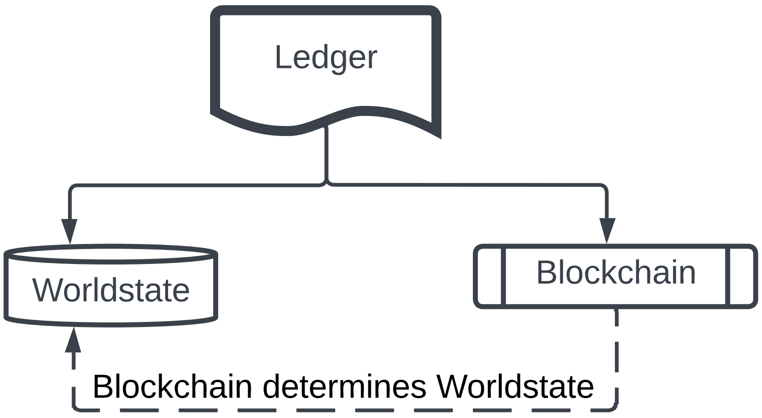


Figure 3. Components of a Ledger.

"12.12",... and at version 0.

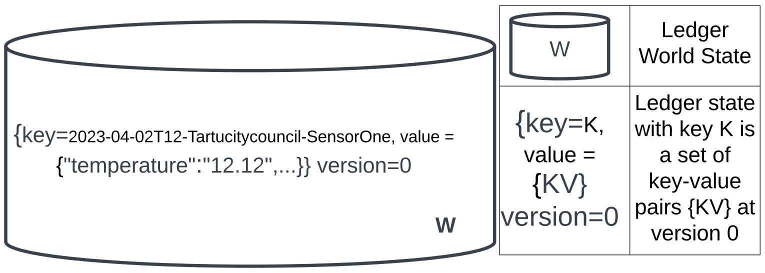

{key=2023-04-02T12-Tartucitycouncil-SensorOne, value = {"temperature":"12.12",...}} version=0
W
W
Ledger World State
{key=K, value = {KV} version=0
Ledger state with key K is a set of key-value pairs {KV} at version 0

Figure 4. Components of a Ledger's Worldstate.

The second component of HLF's ledger is the blockchain which is a log of all the transactions that have led to the world's current status [12]. The blockchain is a series of interconnected blocks arranged sequentially, with each block containing a set of transactions that either update or query the current state of the world. It is worth noting that the ordering of blocks and transactions within blocks is established by the ordering service component of HLF during the initial creation of blocks. The header of each block contains a hash of the block's transactions and a hash of the previous block's header, which links all transactions together securely and sequentially [12]. This cryptographic linking and hashing mechanism ensure the integrity of the ledger data. Even if one node in the network were to be tampered with, it would be unable to convince the other independent nodes that its version of the blockchain is correct, as the ledger is distributed across the network. By examining the blocks of transactions appended to the blockchain, the history of changes that led to the world's current state can be understood.

The blockchain data structure is immutable once created, which is different from the current state of the world. Therefore, the blockchain is a chronological record of the events leading up to the current status of items. Every prior iteration of each ledger state, including any modifications, is recorded on the blockchain [12].

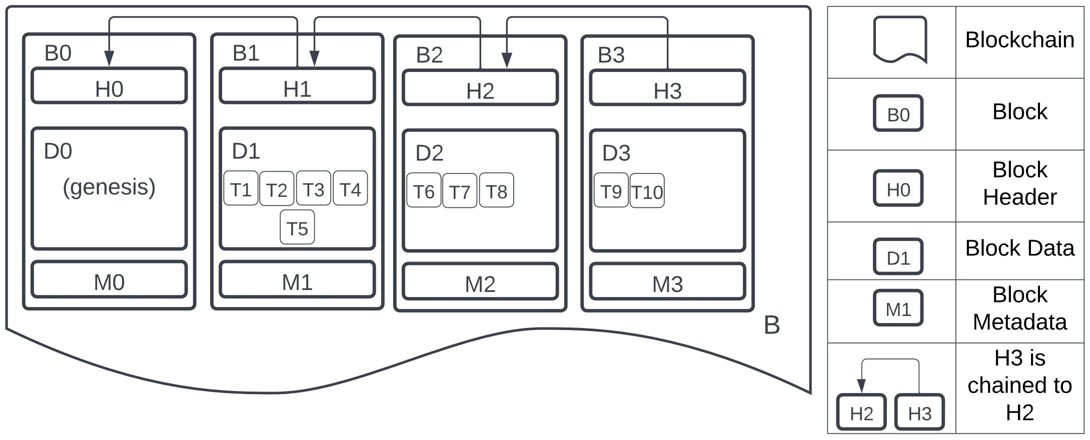


Figure 5. Components of a Ledger's Blockchain.

Unlike the global state, which uses a database, the blockchain is consistently implemented as a file. The components of a ledger's blockchain are presented in Figure 5, which includes a Block, a block header, block data, and block metadata. The figure also depicts the connections between each preceding block and the next block in the chain.

2.5.4 Channels

On an HLF network, channels organize the network participants into subgroups that can collaborate [14]. Channels allow a subset of organizations, peers, and applications to form a private and secure network to conduct transactions and share data. Channels are similar to friendship groups, where individuals can belong to multiple groups with different activities and expectations. Similarly, organizations can belong to other channels with various transactional activities and data-sharing requirements. Each channel has policies and rules for membership, endorsement, and access control.

In an HLF network, the channel is responsible for maintaining a copy of the ledger that its members share. The peers within a channel collaborate to validate and endorse transactions before adding them to the shared ledger. The orderer nodes, which operate outside any channel, receive and order transactions from their peers and distribute them to the appropriate channels. When a chain code is instantiated and deployed to a channel, it becomes available to all the applications authorized to access that channel allowing the applications to interact with the smart contracts in the chain code and perform transactions on the channel's ledger [14].

Channels provide a way to partition an HLF network to limit the scope of communication and data sharing between network participants. By separating the network into smaller, more manageable groups, channels can improve the network's scalability, privacy, and performance. In Figure 6, two separate channels (e.g., Channel 1 and Channel 2) are depicted, each with a single peer belonging to two different organizations, Organization One and Organization Two.

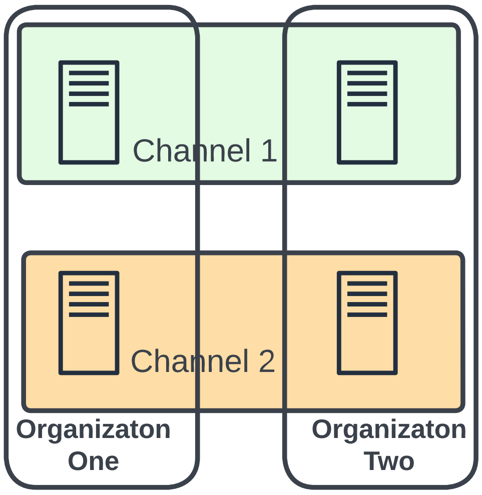


Figure 6. Two Organizations with single Peer in two separate Channels.

2.5.5 Chaincode

The business logic of the transactions is specified by smart contracts, which specifically manage the business entity's life cycle and are incorporated in the world state bundled into chain code [9]. It is then replicated across the whole blockchain network. As a result, smart contracts are specified by chain code. Several smart contracts can be specified within a single chain code. Upon installing the chain code, the application can utilize all the smart contracts available inside that chain code. Chaincode is a container for installing and instantiating numerous comparable smart contracts, whereas smart contracts are customized to the application used to enable business activities [9].

Every chain code has an endorsement policy that applies to any relevant smart contracts that are connected to it. This defines which organizations must sign a smart contract-generated transaction to be considered legitimate.

2.5.6 Transactions

Transactions in HLF are a crucial aspect of the platform, as they enable the exchange of assets and data between participants in the network. In HLF, a transaction is a series of operations by one or more network users. These operations include moving assets, altering the ledger's state, carrying out smart contracts (chain code), and creating events. Transactions in HLF are executed using a consensus mechanism that ensures that all network participants agree on the ledger's state.

In edge computing, a transaction refers to a data exchange or communication between two or more computing devices or entities. These transactions can include various data types, such as messages, files, commands, or queries. In edge computing, transactions occur between edge devices, edge nodes, and the cloud, where each entity can perform specific functions, such as data processing, storage, and analysis. Transactions in edge computing are critical for enabling real-time communication and decision-making and ensuring secure and reliable data exchange between different entities in the network. They also play a vital role in optimizing the performance and efficiency of edge computing systems.

Within the framework of this thesis, a transaction is defined as the complete process of transferring data from the sensors to the external data repository. Specifically, a transaction begins with sensor data generation and concludes once that data is successfully persisted in the external data repository. Several steps/phases are involved in the HLF transaction execution process:

1. Proposal: The client application produces a transaction proposal during this step that contains information on the transaction's specifics, including the asset to be transferred, the amount, and the intended receiver.

2. Endorsement: The request is sent to one or more network peer nodes for approval. Verifying a transaction proposal and confirming that the smart contract's rules permit the suggested activities is called an endorsement.

3. Ordering: Upon endorsement, the transaction is sent to the ordering service, which organizes the transactions and builds a block.

4. Validation: The block is subsequently broadcast to the network's peer nodes for validation. Each peer node verifies the block's transactions and ensures accuracy, given the ledger's current state.

5. Commitment: The block is committed to the ledger, and its state is updated once a sufficient number of peer nodes have validated it.

2.6 Related Works

Blockchain technology has many applications in securing IoT devices in wireless sensor networks. It can be utilized for access control, authorization, authentication, detection of malicious devices or Distributed Denial of Service(DDOS) attacks, and secure storage of IoT data. Although a few existing solutions attempt to address each problem independently, this section focuses on presenting solutions that ensure the secure collection and storage of IoT data. These solutions utilize blockchain technology with all its inherent features, including consensus mechanisms, smart contracts, decentralization, pseudo-anonymity, and immutability.

Shafagh et al. (in [25]) introduced a blockchain-based system that offers distributed access control and data management. The system's contributions include a secure cryptographic method for sharing data with frequent key updates, the ability to revoke access to data, an efficient search method for compressed chunked data streams, and a location-aware level of data storage. However, the architecture's consensus mechanism, the Proof of Work (PoW) mechanism, has a drawback. PoW is resource-intensive and vulnerable to the 51% attack, which may happen if the proposed architecture lacks sufficient users and hash power.

Ren et al. (in [22]) have presented an architecture for the secure storage of data generated by edge devices. Their proposed architecture utilizes blockchain technology, where two blockchain networks are implemented: local and global. The local network, which has limited storage space, is established by the primary edge nodes to store all data generated by IoT devices. On the other hand, the global blockchain network is built using cloud servers and stores all data obtained from local blockchain networks. The cloud servers compute hash values for the uploaded data to the global blockchain, and to ensure data integrity, periodic checks are carried out by comparing the already calculated hash values.

Liu et al. (in [17]) introduce a Data Integrity as a Service (DIaaS) framework that utilizes blockchain technology to authenticate data generated by IoT devices. The framework aims to solve two issues: firstly, eliminating the need for trust in third-party auditors and improving the reliability of the data integration service. Secondly, the framework proposes protocols for verifying data integrity in a fully decentralized environment without relying on a single third-party auditor. However, the main drawback of this approach is that the integrity of the data packets requires validation from a trusted third-party authority.

Ruonan et al. (in [16]) present a scheme that employs blockchain technology to secure storage and protect large amounts of data generated by IoT devices. Their proposed approach ensures data protection by leveraging many blockchain miners to manage IoT device data, eliminating the need for centralized servers. The architecture also adopts edge computing to process and transmit the data to the Distributed Hash Tables (DHT). Additionally, the authors propose using certificate-less cryptography, which reduces the redundancy caused by traditional Public Key Infrastructure (PKI) and provides an efficient authentication method for IoT devices. However, the scheme may encounter issues if it requires the implementation of more complex access control policies.

Sun et al. (in [24]) proposed an architecture consisting of three layers: data access, service, and application. The data access layer collects data through various protocols, and the data service layer stores, shares, and analyzes data using blockchain technology for secure storage. The data application layer provides APIs for device and user management, data sharing, and querying. However, the paper did not address sensor authentication, and storing sensor data on the blockchain can result in high storage costs. Querying the blockchain can also affect performance in large networks.

Our blockchain architecture differs from the ones mentioned earlier by accommodating other IoT devices without blockchain capabilities but who desire to utilize the proposed ecosystem for secure and traceable data storage. These IoT devices perform all verification processes of the proposed architecture, communicating with a Blockchain network deployed in a fog environment near the sensors' location. Moreover, our architecture's configurable authentication timeout enhances overall security. Additionally, we propose storing sensor data off-chain on a more efficient object storage server, reducing the need to query this data from the blockchain network. We do not store the sensor data on the blockchain; instead, we store the hash of the sensor data, minimizing the overall storage cost.

In conclusion, this Section 2 provided an overview of Edge, Fog, and Cloud Computing, highlighting their similarities and differences. Additionally, we explored the concept of Blockchain technology and compared two popular frameworks, HLF and Corda. Our justification for selecting HLF as the reference framework for our proposed architecture was also included. We then delved into the critical components of HLF in detail. The section concluded with a literature review of related works, identifying gaps our research aims to address.

3 Blockchain for Edge to Cloud Continuum

This section introduces our proposed architecture called E2C-Block (Blockchain for Edge to Cloud Continuum) and discusses its design. Additionally, we present a hypothetical scenario where this model architecture can be applied. E2C-Block is a proposed model architecture that aims to provide an effective solution for managing and securing data generated by IoT sensors in a distributed environment. It ensures that data generated by these IoT sensors are securely transmitted and stored tamper-proof. The model comprises two blockchain networks and an external data repository.

The first blockchain network is located in a fog computing environment close enough to the IoT sensors generating the data. This Blockchain is called the Sensor Blockchain Network(SBN). In the E2C-Block architecture, the fog computing environment provides a secure communication channel between IoT devices and the cloud infrastructure. This SBN has peers that process and transmits data to a second blockchain network in a cloud computing environment. The second blockchain network is the larger of the two blockchain networks. It is located in a cloud computing environment. It is called the Primary Blockchain Network(PBN). It receives IoT Sensor data from the SBN, hashes and stores this hash of the sensor data on the ledger of its peers, and passes on the originally received IoT sensor data to an external data repository. As discussed in Section 3, this hashing process becomes useful when we check the integrity of the IoT's Sensors Data. To store the vast amount of data generated by IoT sensors, the E2C-Block architecture uses an offsite data store. This provides an optimal storage option as it allows data to be stored in a secure and scalable manner. Using an external data repository also ensures that data can be accessed and queried while maintaining data privacy and security. This external data repository only receives IoT Sensor data from the PBN. This data stored in this external store is the unhashed, original version of the IoT Sensor Data. So, in essence, very briefly, in E2C-Block ,

1. Multiple sensors generate data sent to the SBN.
2. The SBN receives sensor data and transmits it to the PBN.
3. The PBN only accepts data from the SBN. It hashes and stores the hash of the IoT sensor data, thereafter it forwards the originally received sensor data to an external data repository.
4. The Data Repository is a storage location for all sensor data. It only receives data from the PBN and stores it as is.

Figure 7 provides a high-level overview of E2C-Block , depicting the flow of sensor data through various computing environments. The image shows that the sensor data arrives at the SBN in the fog environment, then moves to the PBN, and finally ends up in the External Data Repository, both located in a cloud environment. Section 4 discusses in more detail what happens at each of the components of E2C-Block .

3.1 Hypothetical Scenario

We describe a fictional use case below to help understand a potential use case of the E2C-Block architecture. The Tartu City Council[1] is interested in collecting, collaborating, and analyzing data from various Internet of Things (IoT) sensors. As the city council cannot provide and operate all these sensors, it has to allow sensors from other organizations to be part of this "network" of sensors. As such, other agencies such as Tartu Transport

[1]This is entirely fictional with no reference to the entities mentioned

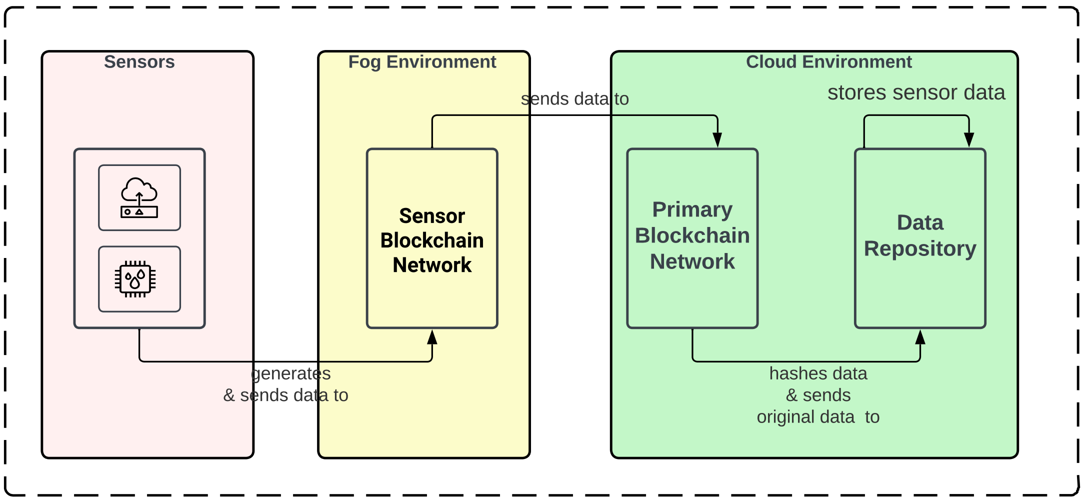

Sensors
Fog Environment
Cloud Environment
sends data to
stores sensor data
Sensor Blockchain Network
Primary Blockchain Network
Data Repository
generates & sends data to
hashes data & sends original data to

Figure 7. High level Overview of E2C-Block .

Service, Tartu Solar Panel Center, Tartu Meteorological Centre, and Tartu Temperature Monitoring Center can bring their sensors to this network. The alliance benefits them as each organization can run analytics on the data their collective sensor pool generates. However, since the data is sensitive, it is crucial to have a secure system to process and transmit the IoT sensor data generated by the sensors to some data repository in the cloud.

After this IoT Sensor data is securely stored, there is also a requirement where each of these participating organizations can query for data stored. However, the proper organization must have access to the data they are only allowed to access or need, and a consistent and immutable log of access is thus required. Finally, data should be almost tamper-proof once stored, as a change or unauthorized access would lead to privacy rights violations. The council has suggested using blockchain technology to add more security to these data as an inherent characteristic of blockchain technology means that data cannot be written or read without the approval of all participating peers in this "network."

If Tartu City Council decides not to use blockchain technology, the transmission of IoT sensor data from different sources and third-party sensors to the cloud could be jeopardized. Without blockchain, the data could be subject to tampering and unauthorized access, creating significant risks. For example, a malicious individual could tamper with the data before it reaches the cloud, compromising its integrity and accuracy. This could result in incorrect analysis and impaired decision-making. Moreover, unauthorized access to sensitive and personal user data in IoT data could lead to privacy violations and potential legal consequences. Even when the data is at rest, there is a risk of unauthorized changes by malicious entities.

3.2 Sensors

The data flow in E2C-Block begins with the IoT sensors. These sensors record and transmit their data to the Sensors Blockchain Network. Sensors can be owned and managed by multiple organizations, and as such, they should be identifiable as belonging to the owning organization. Therefore, each sensor in the E2C-Block architecture can be assigned a unique ID that links it with its organization. The sensors have various configurable properties, including but not limited to Transmit_Int and Sensor_type, that impact their setup and functionality. For more detailed information on these properties, please refer to Table 5.

In addition, E2C-Block allows administrators of owning organizations to add or remove sensors without compromising security. These sensors can authenticate and be registered on the SBN and can generate and transmit a continuous stream of data to the network. We used Python scripts to mimic data generation to implement the sensors' functionality. These Python scripts allow the sensors to be completely configurable, enabling us to set the UniqueId, the send Interval, and the type of sensor data the sensor should generate. The choice of Python was based on providing libraries out of the box that allowed these requirements to be quickly fulfilled. We considered implementing the sensors as Node.js scripts but found it difficult, as Node.js is a single-threaded platform, and it was easier to spawn up multiple threads to generate and send this data with Python.

The generated sensor data is in JSON format. We considered generating and transmitting it in YAML or XML. However, we settled for the JSON format as it is more generally acceptable and easier to be consumed by the smart contracts running on the SBN. Also, generating the same data payload in JSON would always be lighter than a similar payload in XML. These Python scripts can be executed from the command line, and once executed, they start generating a constant stream of data. Code listing 1 illustrates the structure of the data payload typically generated by a sensor. The attributes in this data payload are explained in more detail in Table 4

```
payload = {
  temperature: 12.22,
  timestamp: '2023-04-02T12:00:00.000Z',
  org: 'Tartucitycouncil',
  device: 'SensorOne',
  id: 'a1b2c3d4-e5f6-4b7c-8d9e-0123456789ab'
};
```

Listing 1. Sample Payload from Sensor

Table 4. Sensor Payload Fields.

Field	Description	Restrictions
timestamp	The date and time when the payload was generated	ISO 8601 format
org	The organization that owns the sensor	String
device	The device identifier of the sensor	String
id	The unique identifier of the payload	UUID
arrivalTime	The date and time when the payload arrived at the sensor node	ISO 8601 format
departTimeFromFogNode	The date and time when the payload departed from the fog node	ISO 8601 format
arrivalTimeFromFognode	The date and time when the payload arrived at the blockchain node	ISO 8601 format
departureTimeFromPrimaryBlockchain	The date and time when the payload was fully processed by the blockchain node	ISO 8601 format

3.3 Sensor Blockchain Network

The SBN is the smaller of the two blockchains in the E2C-Block and is architecturally located in the fog computing environment close enough to the IoT Sensors. Its primary function is to act as an intermediary between the IoT sensors and the PBN, transmitting sensor data to the PBN. The SBN is the only component of E2C-Block to communicate with the PBN. Rather than storing data on the ledger of its peers, the SBN consistently listens for a constant stream of sensor data generated by the IoT sensors. It also serves as an authentication and registration point for all sensors, ensuring they are authenticated and registered on the SBN before transmitting data. Communication between the IoT sensors and the SBN is supported via the HTTPS protocol. Additionally, the SBN modifies the received sensor payload by adding two attributed arrivalTime and departTimeFromFogNode. The arrivalTime holds the time the Sensor Payload arrived at the SBN, while the departTimeFromFogNode attribute contains the time this particular sensor payload left the SBN. These attributes would be useful when benchmarking a single sensor data point's time to move to the external data repository. Listing 2 shows

Table 5. Configuration Properties for Sensors.

Configuration	**Description**	**Restricted Values**
HOST	Specifies the Host where the data generated by this sensor would be posted to	User Defined
PORT	Specifies the Port on the Host where generated data would be POSTed to	User Defined
PROTOCOL	Specifies the type of protocol to use in sending the Data	HTTP or HTTPS
ENDPOINT	Specifies the endpoint on the Host where the data would be sent to	Provided by SBN
TOKEN_ENDPOINT	Specifies where this sensor MUST register to and authenticate with	Provided by SBN
USERID	Specifies USERID needed to be authenticated with the SBN	Provided by SBN
USER_SECRET	Specifies USER_SECRET needed to authenticated with the SBN	Provided by SBN
TRANSMIT_INT	Specifies how frequently this sensor should send data	one_second, five_seconds, ten_seconds, fifteen_seconds, twenty_seconds, one_minute, one_hour, one_day
SENSOR_ORG	Specifies the type of data this sensor should generate	Restricted set of values
SENSOR_TYPE	Specifies the type of data this sensor should generate	Humidity, Temperature
SENSOR_ID	This Specifies the unique Id of the Sensory, User Defined	

the modified Sensor Payload before leaving the SBN.

A blockchain network was chosen over a single or server cluster as a proxy due to the inherent benefits a blockchain network provides, such as ensuring that all peers in the network must agree on the action before any sensor is authenticated or registered. This limits the possibility of a rogue and compromised proxy server, allowing an equally rogue and compromised sensor to transmit data. Pushing the authentication and sensor registration workload to the PBN would have been an unnecessary overhead incurred by the PBN. Separating the two blockchain networks allows the SBN to be placed in the fog computing environment, which is naturally closer to the IoT sensor devices. In addition to authenticating and registering IoT sensors, the SBN also initiates a host-level IP blocking of sensors that repeatedly fail authentication.

HLF was chosen as the reference implementation for the SBN due to its features and advantages, which are discussed in subsection 2.4.4.

```
payload = {
  temperature: 12.22,
  timestamp: '2023-04-02T12:00:00.000Z',
  org: 'Tartucitycouncil',
  device: 'SensorOne',
  id: 'a1b2c3d4-e5f6-4b7c-8d9e-0123456789ab',
  arrivalTime: '2023-04-02T12:00:01.000Z',
  departTimeFromFogNode: '2023-04-02T12:00:03.000Z',
};
```

Listing 2. SBN Modified Payload

3.4 Primary Blockchain Network

The PBN is the larger of the two blockchains in E2C-Block . It receives IoT sensor data from the Sensor's blockchain network. In this Blockchain network, The sensor data Payload received from the SBN is modified by adding two extra attributes - arrivalTimeFromFognode and departureTimeFromPrimaryBlockchain - before its hashed value is stored on the ledger of its peers in the PBN. arrivalTimeFromfognode indicates the time the payload arrived from the SBN, while departureTimeFromPrimaryBlockchain records the time the payload left the PBN for the external data source. Listing 3 shows this modified sensor payload.

```
payload = {
  temperature: 12.22,
  timestamp: '2023-04-02T12:00:00.000Z',
  org: 'Tartucitycouncil',
  device: 'SensorOne',
  id: 'a1b2c3d4-e5f6-4b7c-8d9e-0123456789ab',
  arrivalTime: '2023-04-02T12:00:01.000Z',
  departTimeFromFogNode: '2023-04-02T12:00:03.000Z',
  arrivalTimeFromfognode: '2023-04-02T12:00:03.023Z'
  departureTimeFromPrimaryBlockchain: '2023-04-02T12:00:03.027Z'

};
```

Listing 3. PBN modified payload

Before storing the modified Sensor Payload on the ledger of its peers, the PBN hashes the data using a chosen hashing algorithm. Hashing involves taking an input message of any length and generating a fixed-size output hash of a specific size. The chosen hashing algorithm is SHA-256 [2], which produces a fixed-size output of 256 bits or 32 bytes. SHA-256 is widely used and has been benchmarked as both performant and secure.

Hashing is a one-way mathematical operation, and the resulting hashed value is typically a unique representation of the original input. Even a small change in the input data will result in a significantly different hash value. For this reason, hashing is useful for ensuring data integrity and authenticity.

We also considered other hashing algorithms, such as Blake2 [3] and MD5. Blake2 performed slightly better than SHA-256 in published benchmarks [4], but SHA-256 was chosen due to its widespread adoption, security, and performance. MD5 was not considered due to vulnerabilities discovered in the algorithm over time.

After hashing the data, the unhashed Sensor Payload is transmitted to the External Data Repository. Figure 8 depicts the data flow from the sensors to the External Data Repository. It shows that the sensor sends an authentication request to the SBN and sends a continuous data stream upon successful authentication. The figure also illustrates the SBN sending an authentication request to the PBN. Upon successful authentication, the modified payload is hashed and stored on the ledger before being sent to the MinIO storage server. HLF was also chosen as the reference implementation for the PBN due to its features and advantages, which are discussed in subsection 2.4.4.

3.5 External Data Repository

The external data repository is the last component of the E2C-Block . It receives IoT Sensor data from the PBN and stores these unhashed sensor data.

Buckets are created for each organization's sensor, and data from that particular sensor is stored there. This ensures that data is well organized. This external data repository would be the primary query point for all sensor data. However, at configurable intervals, when a particular sensor data is queried (with the unique Sensor Id, e.g., a1b2c3d4-e5f6-4b7c-8d9e-0123456789a), the authenticity of the data can be verified from the PBN via an HTTP request. The arguments for this request would be the hashed value of this sensor's Id Payload stored in the external data repository. We then compare the new hash with the previously stored hash on the PBN. If these hashes are the same, the data has not been tampered with since it came to rest on the external data repository. If the hashes are different, the data has been tampered with.

To handle large amounts of IoT sensor data with no real relationships, we chose to use an object storage server instead of a relational database management system (RDBMS) which is not designed to scale horizontally. Object storage servers are optimized for

[2]https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf/

[3]https://www.blake2.net/

[4]https://public-inbox.org/git/20180609224913.GC38834@genre.crustytoothpaste.net/

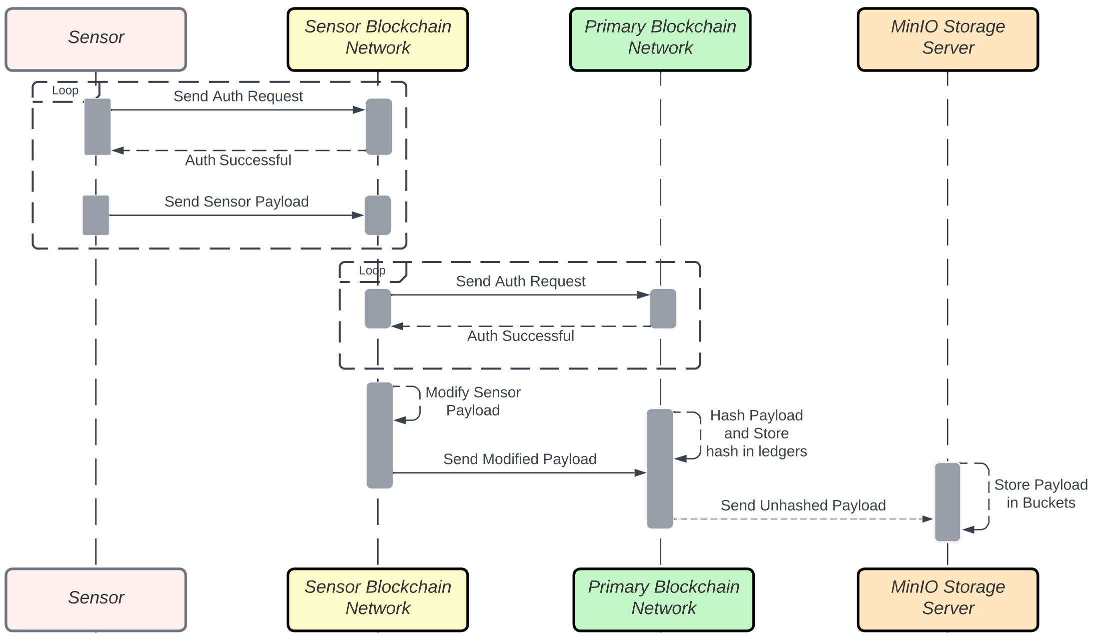


Figure 8. Data flow from Sensors to MinIO.

extensive data processing, making them a better option for an external data repository.

We evaluated MinIO, Amazon S3 [5] and Ceph [6] as possible candidates for an Object Storage server. MinIO was selected as an option for the external data repository. MinIO is known for its ability to handle large data quantities, making it a reliable and efficient choice. It offers robust capabilities in handling large data quantities and can easily be deployed on-premises or in the cloud. It provides a simple yet powerful interface for object storage and offers features such as access control, versioning, and lifecycle policies that can be used to manage the data effectively. Additionally, MinIO is open-source and can be customized to meet specific requirements, making it a flexible solution.

MinIO is a popular open-source object storage system compatible with Amazon S3 cloud storage service. It provides a scalable and distributed storage solution for unstructured data such as images, videos, and documents. MinIO is designed to run on commodity hardware and is optimized for performance, making it ideal for use cases where fast and efficient data access is critical.

One alternative was Amazon S3, a cloud-based storage service that Amazon Web Services (AWS) provides. AWS is a well-established cloud computing platform with many services, including S3. S3 is highly scalable, durable, and secure, making it reliable

[5]https://aws.amazon.com/s3/
[6]https://ceph.io/

for storing large amounts of data. Additionally, it offers a range of features such as versioning, access control, and lifecycle policies that can be used to manage the data effectively. However, using S3 would require us to use AWS, which was not feasible for our use case. Another alternative that was considered was Ceph, an open-source distributed storage system. Ceph provides highly scalable and fault-tolerant storage, making it a suitable option for storing large amounts of data. It offers features such as object storage, block storage, and file storage, providing a flexible storage solution that can be customized to meet specific requirements. However, setting up and configuring Ceph can be complex and require significant time and resources to deploy and maintain.

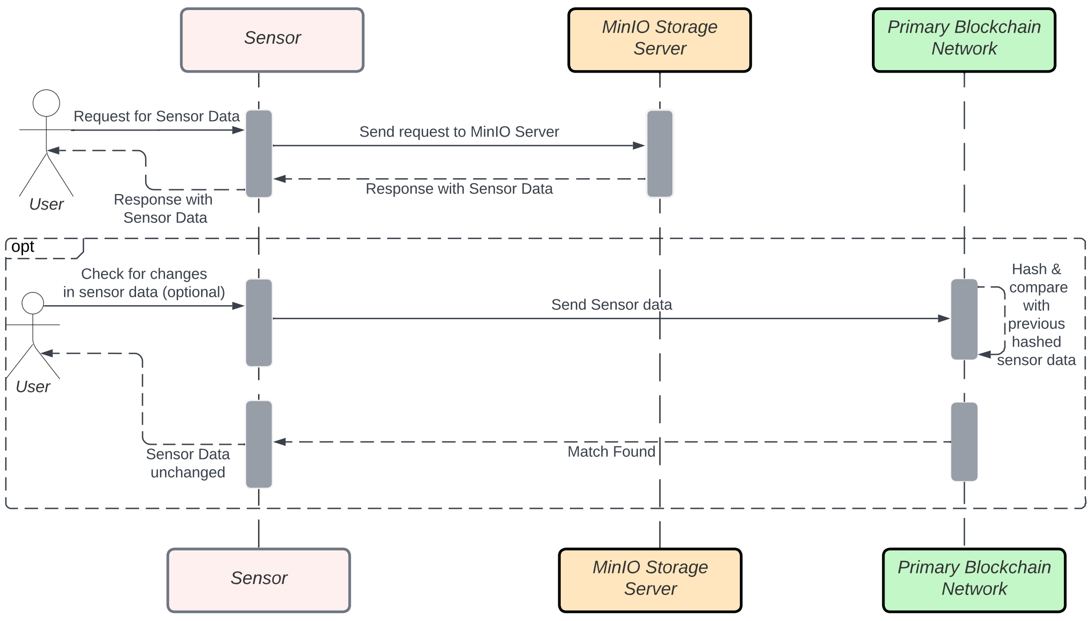
3.6 How is Stored Sensor Data Queried?

We developed an interface [7] for browsing the stored data to enable querying Sensor data stored at rest on the MinIO Storage server. The interface loads all available buckets and their respective sensor data. Each bucket is mapped to an individual sensor owned by different organizations. This interface is a read-only interface that does not allow modification of already stored sensor data. When a user clicks on a particular sensor detail, the interface displays the details of that specific sensor reading, including the last time the payload details were verified as authentic from the PBN.

Additionally, the interface features a verification button that instantly confirms if the details of the sensor payload match those on the PBN. Upon clicking this button, a request is sent to the PBN with the specific sensor data, which is hashed with the same function used to hash the original sensor data. If the hashes match, the data has not been altered. Figure 9 shows a diagrammatic flow for requests to query for sensor data, as previously described.

In conclusion, in Section 3, we introduced E2C-Block and a hypothetical scenario for its application. The architecture design, from sensors to the External Data Repository, was thoroughly discussed, with justifications for each decision. This section answers the first research question of best security measures for edge computing environments by proposing using the SBN in the fog environment, which provides an additional layer of security for IoT sensors at the edge.

[7]https://github.com/chinmaya-dehury/Blockchain4E2CC



Sensor
MinIO Storage Server
Primary Blockchain Network
User
Request for Sensor Data
Send request to MinIO Server
Response with Sensor Data
Response with Sensor Data
opt
Check for changes in sensor data (optional)
Send Sensor data
Hash & compare with previous hashed sensor data
User
Sensor Data unchanged
Match Found
Sensor
MinIO Storage Server
Primary Blockchain Network

Figure 9. Flow for Querying Sensor Data from MinIO Storage.

4 Implementation

This section covers the implementation specifics of E2C-Block , starting with the Sensors and progressing through the SBN, the PBN, and ending with the external data repository, MinIO. The code snippets presented throughout this section and section 5 have been tested and deployed and can be found at the https://github.com/chinmaya-dehury/Blockchain4E2CC repository.

4.1 Sensors

E2C-Block begins with IoT Sensors generating and transmitting data. However, before these sensors transmit this generated data to the SBN, they must authenticate with the blockchain network. They must have been previously registered on the SBN.

4.1.1 Authentication and Registration

Authenticating and Registering sensors on the SBN is essential in ensuring the security and reliability of data transmitted through E2C-Block . The first step in this process is for the sensors to make a POST authentication request to an endpoint specified in TOKEN_ENDPOINT, located within the SBN. For this authentication request, the sensors must provide their credentials, including their USERID and USER_SECRET values. Once the TOKEN_ENDPOINT on the SBN receives the authentication POST request from the sensors, the provided credentials are verified to ensure that the requesting Sensor is authorized to transmit data over the network.

If the provided credentials are valid, the requesting Sensor is granted permission to transmit data to the SBN. If authentication fails, and there are subsequent failed multiple authentication requests, the SBN adds the originating IP address for these failed requests to the UFW firewall Table. Subsequent authentication requests from this Sensor are thus dropped at the OS level without getting to the Blockchain Network. This authentication process is made possible by the Fabric Certificate Authority (CA) server, a vital HLF blockchain framework component. The details of how the HLF Certificate Authority does these are discussed in subsection 4.2.

Another essential aspect of the authentication process is that there exists an authentication window in the SBN. The authentication window is a customizable feature with a preset default value of 10 minutes. Within this timeframe, the Sensor can communicate with the SBN without further authentication. However, once the authentication window has elapsed, the Sensor must reauthenticate to continue communicating with the SBN. This mechanism serves as a safeguard to ensure that only authorized sensors can access and interact with the SBN, bolstering the overall security and integrity of the system. During the initial authentication request, the sensors also provide their SENSOR_ID and SENSOR_ORG values, which verify that the Sensor has been added to the SBN's ledger. This step ensures that only authorized sensors can transmit data over the network.

While the Sensor initiates authentication, Sensor registration is initiated on the SBN. Sensor Registration means an authorized network administrator adds the SENSOR_ID to the SBN. This is a one-time process. At any point, a network administrator can also de-register a sensor. Subsection 4.2 details the Sensors registration process.

Overall, authentication and registration are crucial to ensuring the security and reliability of data transmitted through E2C-Block .

4.1.2 Data Generation and Transmission

Once sensors have been authenticated and their registration status confirmed on the SBN, the next step involves generating and transmitting data to the SBN.

The Python script initiates data generation at predetermined intervals, as specified by the TRANSMIT_INT configuration value. This data is transmitted via the protocol specified in the PROTOCOL configuration value, with HTTPS being the typical choice for secure data transmission. To send the data, an HTTP POST request is made to an endpoint that combines the HOST, PORT, and ENDPOINT configuration values. This data transmission process will continue until the Python script is terminated, ensuring that the SBN receives a consistent data flow from all registered sensors.

The Sensor's Python script has an exponential backoff mechanism that retires the transmission request in the improbable event that the SBN is overloaded for any reason.

On the SBN, a smart contract continually listens for incoming data from all authenticated and registered sensors and processes it accordingly.

Although the sensor authentication, registration, data generation, and transmission phases may seem distinct, they are part of a single process that runs automatically and continuously when executing the Sensor Python Script. Listing 4 illustrates the command responsible for generating sensor data for Sensor One of the Tartu City Council.

```
cd sensors/tartucitycouncil/sensor-one
flask --app app.py run
```

Listing 4. Generating sensor data

4.2 Sensor Blockchain Network

The SBN plays a crucial role in E2C-Block by providing an additional layer of security and authenticity to the data collected by the IoT sensors. By serving as an intermediary between the sensors and the PBN, the SBN can authenticate and register sensors, ensuring that only authorized sensors can transmit data. Additionally, the SBN can verify the authenticity of sensor data before sending it to the PBN, reducing the load on the network and ensuring that only validated data is stored.

One of the key benefits of using a Blockchain network in the Fog computing environment is the ability to leverage the distributed consensus mechanism inherent in blockchain technology. By having multiple peers on the network validate each sensor and data point, the authenticity of the data is better controlled, and the risk of a single point of failure is greatly reduced. This is in contrast to a conventional fog node, which may be vulnerable to rogue actors compromising the data it transmits to the PBN.

This SBN comprises two peers and a single Solo orderer. These peers are all in the same single channel. It also has a chain code installed on it that allows it to carry out its two major functions of

1. Authenticating and Registering Sensors

2. Transmission of data to the PBN.

4.2.1 Sensor Authentication and Registration

Authentication of sensors on the SBN is a crucial step in ensuring the integrity and security of the network. The process of authentication is made possible by the Fabric Certificate Authority (CA) server, a vital component of the HLF framework. The HLF CA server generates and manages digital certificates for authentication and authorization, providing a trusted third-party certificate authority for the Fabric network.

When a sensor is enrolled in the HLF network, the HLF CA server generates an X.509 certificate and a private key for the sensor, which is used to establish secure communication channels with the network. The certificate contains the sensor's identity information, such as its name, public key, and other relevant metadata. On the other hand, the private key is used to sign transactions and verify the sensor's identity during communication with the network.

To authenticate a sensor with the HLF CA server, the sensor must first be enrolled in the Fabric network. Enrolling a sensor involves sending a certificate signing request (CSR) to the HLF CA server, which verifies the sensor's identity and generates an X.509 certificate and private key. The sensor can then use these credentials to interact securely with the network. Authenticating a sensor on the SBN in E2C-Block and obtaining the necessary credentials was simplified using tools like the Hyperledger Fablo Rest API. With this tool, enrolling and authenticating a sensor with the HLF CA Authority Server of the SBN can be achieved through a simple HTTP request to a provided endpoint, passing in the required username and password. Listing 5 shows a code snippet of the Smart contract running on the SBN that tries to authenticate an IoT Sensor.

```
async function getToken() {
  const TOKEN_ENDPOINT = config.TOKEN_ENDPOINT;
  const headers = {
    Authorization: "Bearer ",
    "Content-Type": "application/json",
  };
  const response = await fetch(TOKEN_ENDPOINT, {
    method: "POST",
    headers: headers,
    body: JSON.stringify({ id: "admin", secret: "adminpw" }),
  });
  const data = await response.json();
  return data.token;
}
```

Listing 5. Sensor authentication function

This request returns an authorization token configured to expire every 10 minutes. This token is then passed with every other API request from the Sensor to the SBN.

On the other hand, Sensor Registration is a one-off process. Sensor registration on the SBN is a crucial process that requires a network administrator or someone with administrative privileges. During registration, the administrator sends a unique passphrase and unique SENSOR_ID to the SBN. These values are encrypted and stored in the network peers' ledger, ensuring the data is immutable and tamper-proof. In the SBN, all participating peers must run a consensus on the submitted request for registration before a sensor can be successfully registered. This consensus ensures that the transaction is validated and agreed upon by all peers in the network before it is added to the ledger. This process guarantees the integrity and security of the network and prevents unauthorized access. Listing 6 shows a typical curl request sent to register a sensor on the SBN.

```
  curl --location 'http://127.0.0.1:8801/invoke/fognodechannel/
     fognode' \
--header 'Authorization: auth_token' \
--header 'Content-Type: application/json' \
--data-raw '{
  "method": "KVContract:registerSensor",
  "args": [
    "a1b2c3d4-e5f6-4b7c-8d9e-0123456789a",
    "TartuCityCouncil:sensorOne@ut"
  ]
}
```

Listing 6. Sensor Registration CURL Request

4.2.2 Sensor Data to Primary Blockchain Network

At every point, authenticated and registered IoT sensors transmit data to the SBN. However, the received data is not stored on the SBN. Instead, it is sent to the PBN.

Upon Sensor data arrival to the SBN, a smart contract on the SBN verifies whether the transmitting SENSOR_ID has been previously registered with this Blockchain Network. If the SENSOR_ID has not been registered, the traffic from that sensor is discarded, and the sensor is added to a host-level firewall, blocking any further transmission from that particular IoT sensor. If the IoT Sensor had been previously registered, the smart contract extracts the sensor data payload and adds two new attributes, arrivalTime, and departTimeFromFogNode, to the sensor payload data. The details of these fields are described in Table 4. Once the sensor payload is modified, it can be sent to the PBN. The content of this modified sensor payload was shown in Listing 2.

Before the SBN can connect to the PBN, it must be authenticated. Similar to the Sensors authenticating with the SBN discussed in Section 4.2.1, the HLF Certificate Authority (CA) server handles this authentication process. The authentication request is made to the PBN using the Fablo REST API and the required credentials. If the response is successful, an authentication token is returned, which is used in subsequent requests. Once the SBN is authenticated with the PBN, the data transmission commences. Also of note is that there is a 10 minutes authentication window after which the SBN must re-authenticate with the PBN. A POST request is made to an endpoint exposed by a smart contract running in the PBN to send this modified payload. Listing 7 shows a function in the smart contract that sends the sensor payload to the PBN.

```
 async function sendDatatoBlockchain(data) {
  const PRI_BLOCKCHAIN_ENDPOINT = config.PRI_BLOCKCHAIN_ENDPOINT;
  const headers = {
    Authorization: "Bearer " + (await getToken()),
    "Content-Type": "application/json",
  };
  const response = await fetch(PRI_BLOCKCHAIN_ENDPOINT, {
    method: "POST",
    headers: headers,
    body: JSON.stringify(data),
  });
  return await response.json();
}
```

Listing 7. Sending data to the PBN

4.3 Primary Blockchain Network

The PBN is the larger of the two blockchain networks in E2C-Block . It comprises ten peers, with each of the five participating organizations contributing two peers, and includes a Solo Orderer. It also has six channels, which are private sub-networks that allow a subset of peers to communicate with each other securely. Five of these channels have peers from the same organization as members. In contrast, the remaining channel has all participating peers as members allowing for secure communication and data sharing between peers within the same organization or across all organizations. One of the primary functions of this PBN is to receive and store the hashed value of sensor data from the SBN.

When the PBN receives the data, a smart contract modifies the payload by adding two extra attributes - arrivalTimeToBlockchain and departureTimeFromPrimaryBlockchain. This modified payload on the PBN was shown in Listing 3. After this, the sensor payload is hashed using an SHA-256 hashing function from the Node.js built-in crypto module. The ledgers of the peers of the PBN store this hashed data. Hashing converts data into a fixed-size output, which is unique for each input. This process helps ensure the data's integrity and makes it tamper-evident while ensuring that the stored payload has a reduced size. The PBN forwards the data to an external data repository for offsite storage after storing it in the ledger.

The communication between the PBN and the external data repository - the MinIO storage server is established using MinIO's Javascript SDK, which provides a reliable and secure connection for data transmission. The MinIO SDK utilizes a set of APIs that allows the PBN to access and manage data on the MinIO server.

Moreover, the data transmission process is carried out continuously and asynchronously. This continuous and asynchronous transmission means that data is transmitted between the PBN and the MinIO Storage server without interruption or delay, ensuring that the data is up-to-date and accurate. This asynchronous transmission enables the PBN to continue processing transactions and other tasks without waiting to complete data transmission. Listing 8 shows the smart contract code responsible for sending the sensor data to the MinIO Storage server.

```
async put(ctx, key, data) {
  const hash = hashData(data);
  await ctx.stub.putState(key, hash);
  const minioClient = new Client(process.env.MINIO_URL, process.env.
      MINIO_PORT, process.env.MINIO_ACCESS_KEY, process.env.
      MINIO_SECRET);
  const valueObj = JSON.parse(data);
  const bucketName = `${valueObj.org}${valueObj.device}Bucket`.
      toLowerCase();
  const objectName = `json/${valueObj.timestamp}-${valueObj.org}-${
      valueObj.device}.json`;
  minioClient.putJson(bucketName, objectName, valueObj, err => err &&
       console.log(err));
  return { success: "OK" };
}
```

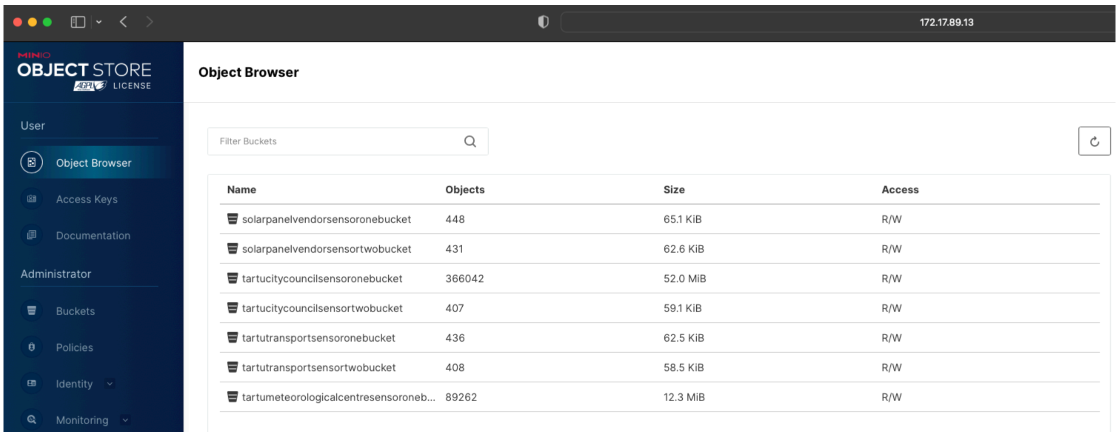
Listing 8. Sending Sensor Payload to MinIO

This PBN also has a smart contract that can be used to verify the authenticity of any previously stored sensor data. It simply hashes the Sensor payload received with the request from the MinIO server with the original hash function and compares that both hashes are equal. If both hashes are equal, the data has not been tampered with after storing it. The response is sent back to the client, querying for the data. Section 3.6 provided some details about this process. It should be noted that the MinIO Storage Server is the primary point of query for all data and not the PBN.

4.4 External Data Repository

The External Data Repository is a critical component of E2C-Block as it is the centralized repository for all generated IoT sensor data. It continuously receives sensor data from the PBN and stores it unhashed as JSON objects in specific buckets. To facilitate data management and access, a unique bucket is created for every sensor belonging to an organization. Figure 10 shows the MinIO interface with some buckets created to store generated sensor data.

The External Data Repository's sole data source is the Primary Sensor Network, which uses its generated MINIO_ACCESS_KEY and MINIO_SECRET to communicate with it. In E2C-Block , we utilize the MinIO Storage Server as the Data Repository to provide a reliable and scalable storage solution. One of the significant advantages of using MinIO is its ability to store data as-is without any further modifications. This means that the JSON Payload received is stored precisely as it was transmitted, ensuring the data's integrity and authenticity throughout the storage process. The MinIO Storage Server is in the cloud environment and runs on Ubuntu 22.04. The MinIO Storage Server serves as the primary query point for the stored sensor data. Therefore, all requests to read sensor data are directed to the MinIO Storage Server.



Name	Objects	Size	Access
solarpanelvendorsensoronebucket	448	65.1 KiB	R/W
solarpanelvendorsensortwobucket	431	62.6 KiB	R/W
tartucitycouncilsensoronebucket	366042	52.0 MiB	R/W
tartucitycouncilsensortwobucket	407	59.1 KiB	R/W
tartutransportsensoronebucket	436	62.5 KiB	R/W
tartutransportsensortwobucket	408	58.5 KiB	R/W
tartumeteorologicalcentresensoroneb...	89262	12.3 MiB	R/W

Figure 10. MinIO Interface showing Sensor Data in buckets.

To optimize the MinIO Storage server, we implemented several optimizations. Firstly, we increased the cache size of the MinIO Server to reduce the frequency of disk I/O operations. This significantly improved the server's response time and reduced the load on the server's hardware. Secondly, we configured the MinIO server to use Direct I/O instead of Buffered I/O. Direct I/O enables the MinIO server to read and write data directly from and to the disk without buffering the data in memory. This helped to reduce the server's memory footprint and improve its overall performance. Lastly, we enabled compression on the MinIO server to reduce the storage space required to store the sensor data. This optimization helped to significantly reduce storage costs, mainly when dealing with large amounts of data.

4.5 E2C-Block Deployment

This subsection briefly describes how the various components of E2C-Block , starting with the Sensors and progressing through the SBN, the PBN, and ending with the external data repository, MinIO were deployed. E2C-Block is mainly made up of two Blockchain networks and a MinIO Storage Server. These Blockchain networks were deployed on Ubuntu 22.02 Servers. We used Ansible [8] to set up these servers and install the prerequisites software packages to automate the server provisioning.

Ansible is a powerful automation tool that provides a server provisioning and configuration management platform. It allows for managing and deploying software applications and infrastructure, making building, configuring, and maintaining servers in large-scale environments easier. As deploying a complete Blockchain network falls outside the expertise of Ansible, we employed Hyperledger Fablo, a specialized tool, to ensure the

[8]https://www.ansible.com/

successful deployment of Blockchain networks in the E2C-Block .

The blockchain networks in E2C-Block were deployed using Hyperledger Fablo. Hyperledger Fablo [9] is a powerful open-source tool that simplifies setting up, deploying, and managing blockchain networks. It allowed us to define the characteristics of these two blockchain networks using JSON files, making it easier than ever to create the desired network topology. With Fablo, we have a flexible configuration file that describes the desired network topology, including multiple organizations, channels, chain codes, and private data collections.

Fablo parses this configuration file into a working HLF Blockchain network. Fablo runs all the components of the fabric network as docker containers, making it easy to manage and scale the network as needed. Fablo supports TLS, RAFT, and solo consensus protocols, and it also exposes a REST API that can be used to interact with the various deployed components of the HLF Network. We used this REST API extensively to communicate with the multiple components of the blockchain network in this thesis. Listing 9 shows the sample command to build an HLF from a fable-config.json [10] file.

```
fablo up fablo-config.json
```

Listing 9. Command to build a HLF network from a config file

Conclusively, Section 4 discussed the implementation of E2C-Block, starting with sensor authentication and data transmission to the SBN. We provided code snippets and diagrams demonstrating how the SBN authenticates, modifies, and sends the payload to the PBN. We also covered the hashing process on the PBN, the data storage on the External Data Repository, and the deployment process, including the tools used for server provisioning and HLF network deployment. This section answered Research Questions Two and Three by showing how blockchain, fog computing, and edge computing can be integrated to reduce storage costs, ensure data integrity, and lower latency. Also, using two blockchain networks decreases latency, and hash-based data storage reduces storage costs. Furthermore, the PBN guarantees data integrity even if data is tampered with on the MinIO server.

5 Experiments

This subsection briefly describes the experiments conducted to benchmark the performance of E2C-Block .

[9] https://labs.hyperledger.org/labs/fablo.html

[10] https://github.com/chinmaya-dehury/Blockchain4E2CC/blob/main/fablo/fablo-config.json

5.1 Experiment Setup

In this thesis, we conducted a benchmarking experiment to evaluate the performance of the PBN in E2C-Block using a workload of transactions generated by a simulation tool. The experiment aimed to measure the network's throughput, latency, and resource utilization under a workload that emulates a real-world scenario. The experimental setup was designed to ensure that the results were reliable, accurate, and reproducible.

5.1.1 Benchmarking Tool

To evaluate the performance of E2C-Block 's PBN, we employed Hyperledger Caliper [11], an open-source benchmarking tool specifically designed to measure the performance of blockchain networks.

5.1.2 Network Configuration

The E2C-Block 's PBN has ten peers and a Solo Orderer. TLS support was not available during the experiments. A channel containing the ten peers was created for testing purposes, and the installed chain code was tested. The chaincode's function was to receive data from Hyperledger Caliper, hash the data, store the hash on the ledger of the PBN's peers, and send the unhashed data to the MinIO Storage server. The batch size was set at 20MB. These details were defined in a network.yml [12] file

5.1.3 Workload Generation

We used the workload module of Caliper to define the smart contract to benchmark on the PBN. These details were defined in a readAsset.js [13] file The workload module interacts with the deployed smart contract during the benchmark round. This module extends the Caliper class WorkloadModuleBase from caliper-core and has three overrides:

1. initializeWorkloadModule - initializes any required items for the benchmark.
2. submitTransaction - interacts with the smart contract method during the monitored phase of the benchmark.
3. cleanupWorkloadModule - cleans up after the completion of the benchmark

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

[12] https://github.com/chinmaya-dehury/Blockchain4E2CC/blob/main/fabric-benchmarks/caliper/networks/networkConfig.yaml

[13] https://github.com/chinmaya-dehury/Blockchain4E2CC/blob/main/fabric-benchmarks/caliper/workload/readAsset.js

5.1.4 Benchmark Configuration

The benchmark configuration file defines the benchmark rounds and references the defined workload module(s). It specifies the number of test workers to use when generating the load, the number of test rounds, the duration of each round, and the rate control applied to the transaction load during each round. Additionally, it includes options relating to monitors. We used the Prometheus monitor to send caliper-related metrics to the Prometheus server. Configurations are defined in a benchmark.yml [14] file.

5.1.5 Hardware and Software Specifications

The virtual machines used to host the blockchain network, simulation tool, and Caliper were running Ubuntu 20.04 LTS. The nodes were configured with 8 vCPUs and 64 GB of RAM, with 50 GB of storage available. The nodes were running HLF v2.4 and Caliper v0.5. The computing resources required for hosting E2C-Block and conducting experiments were provided by the HPC Center [26] at the University of Tartu.

5.2 Performance Metrics

To evaluate the performance of the Primary blockchain network in E2C-Block , we measured the following performance metrics using Hyperledger Caliper:

- Transaction throughput: Transaction throughput refers to the rate at which valid transactions are successfully committed by a blockchain network over a specific period. It is a critical performance metric that measures the efficiency and capacity of the network in processing and validating transactions.

- Transaction Latency: Transaction latency refers to the amount of time it takes for a blockchain network to confirm and commit a transaction, starting from the point at which it is submitted to the point at which it is available across the network. The time delay occurs between initiating a transaction and the network validating and processing it.

- Block Size: The block size refers to the maximum number of transactions that can be included in a single block. The block size in HLF can be configured by adjusting the maximum block size setting in the network configuration file. The maximum block size can impact the network's overall throughput and latency and the time it takes to validate and propagate a new block.

[14]https://github.com/chinmaya-dehury/Blockchain4E2CC/blob/main/fabric-benchmarks/caliper/benchmarks/myAssetBenchmark.yaml

- Block Propagation Time: Block propagation time refers to the time it takes for a newly created block to be disseminated across the network and committed to the ledger by all participating nodes. HLF uses a gossip protocol to disseminate blocks to participating peers to ensure fast block propagation time. This protocol allows nodes to communicate with a subset of other nodes in the network, exchanging information about the blocks they have received.

- Consensus time: Consensus time refers to the time it takes for a blockchain network to reach a consensus on a new block and add it to the blockchain. It is the time it takes for the nodes in a blockchain network to agree on the validity of a new transaction and add it to the blockchain.

5.3 Experiment Execution

We conducted four major Experiments:

- In the first experiment, the objective was to study the impact of block size on the overall network performance. We varied the block size between 10 to 50 transactions while sending transactions at various send rates and collected performance metrics such as throughput, transaction latency, response time, block propagation time, and consensus time for each block size.

- In the second experiment, we focused on the effect of transaction rate on the performance of the PBN. We varied the transaction rate from 100 to 3000 transactions per second.

- In the third experiment, we varied the number of participating nodes in the PBN to study its impact on the network performance, particularly consensus time. Consensus is the process of validating and adding new blocks to the blockchain, and it involves a certain number of nodes agreeing on the validity of a new block. The more nodes are involved in the consensus process, the longer it may take to reach consensus. We varied the number of nodes from 10 to 100 and collected the same performance metrics as in the previous experiments.

- The last experiment measured the time it took for sensor data to travel from the SBN to the PBN and then to the MinIO data repository. This experiment focused on the data transfer aspect of the blockchain network, as opposed to consensus or transaction processing. The sensors generated data at varying rates between 100 and 3000, and the experiment observed the time it took for the data to reach the MinIO server. Its goal was to evaluate the efficiency of the data transfer process in the blockchain network and identify any possible bottlenecks or limitations.

In Section 5, we presented our experimental setup, including the benchmarking tool, network, workload configurations, and hardware and software specifications. We also introduced and explained the performance metrics used to evaluate the results of our experiments. The chapter concludes with a discussion of the major experiments performed.

6 Results and Discussions

The following results were obtained from the four experiments conducted to evaluate the performance of the PBN in E2C-Block .

6.1 Impact of Block Size

Figure 11 illustrates the average throughput across various block sizes at different transaction sending rates. At the same time, Figure 12 shows the average latency over the same rates. The experiment involved sending transactions at speeds ranging from 10 tps to 500 tps, with multiple parameters listed in Table 6, including transaction sending rate, block size, and number of peers.

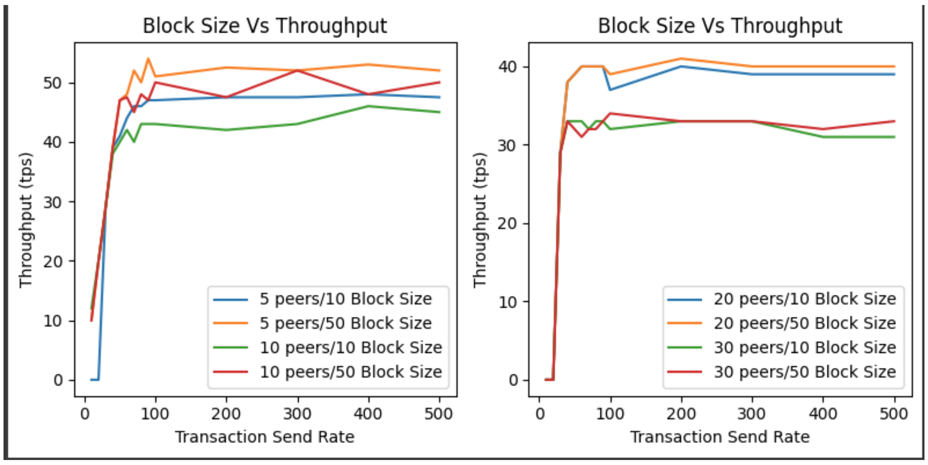


Figure 11. Block Size vs Throughput

Figure 12 shows that the average latency remained under 1 second throughout the experiments until it approached around 100 tps. The system's throughput increased linearly as the transaction sending rate increased, plateauing at approximately 100 tps, indicating the highest usable rate. Beyond this point, the system's performance degraded as the load increased.

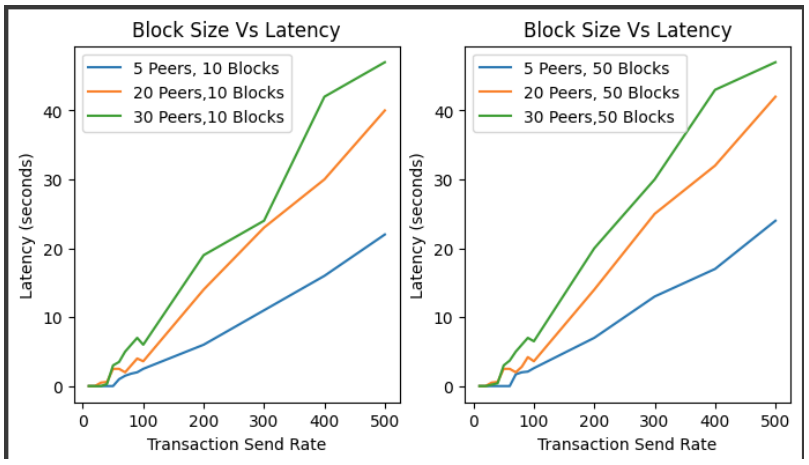


Figure 12. Block Size vs Latency

However, the blockchain system's performance depends on the hardware capabilities of the System under Test and the number of involved peers, which also increases latency. HLF is built on a Docker-based architecture, with each hyper ledger network component running in a separate container communicating via a created network. Using smaller block sizes with low transaction rates was preferred to achieve lower transaction latency in real applications like IoT. Conversely, higher transaction rates required larger block sizes in order to achieve higher throughput and lower transaction latency.

Table 6. Experimental Parameters

Parameters	**Values**
Transactions Sending Rate	10, 20, 30, . . . , 100, . . . , 500 (tps)
Number of Peers	5, 10, 20, 30
Block Size	10, 50

6.2 Impact of Transaction Rates

Figure 13 shows the relationship between Transaction Rates and throughput at various peer sizes. From the figure, we can observe that as the number of peers increases, the throughput decreases. For example, at a transaction send rate of 10, the throughput for five peers is 15, while the throughput for 30 peers is 15. At a transaction send rate of 50, the throughput for five peers is 50, while the throughput for 30 peers is 50. However, at higher transaction send rates, the difference in throughput between different peer sizes becomes more significant.

Another observation is that the throughput sometimes increases as the transaction send rate increases. For instance, at a transaction send rate of 300, ten peers' throughput is higher than five peers. However, for a transaction send rate of 400, five peers' throughput is higher than ten peers. This implies that there may be an optimal transaction send rate for a specific peer size that maximizes the network's throughput. In conclusion, the experiment suggests that increasing the number of peers in a network may only sometimes result in higher throughput. There may be an optimal transaction send rate that maximizes the network's throughput for a particular peer size.

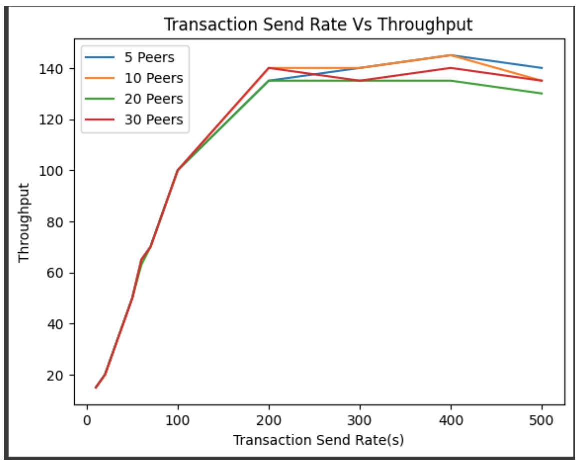


Figure 13. Transaction Send Rate(s) vs Throughput

On the other hand, Figure 14 shows the relationship between Transaction Rate and latency at various peers configurations. From the figure, we can see that the latency is relatively low when there are few peers. For example, when there are only five peers, the latency is between 0.1 to 0.5 seconds. As the number of peers increases, the latency also increases significantly. For instance, when there are 30 peers, the latency can be as high as 1.4 seconds for a transaction send rate of 200.

The latency continues to increase as the number of peers increases, and it can be as high as 9.6 seconds for a transaction send rate of 500 when there are ten peers.The figure also reveals that the transaction send rate significantly impacts the latency. The latency increases as the transaction send rate increases, especially when there are many peers. For example, when there are 30 peers, the latency increases from 1.4 seconds to 6 seconds when the transaction send rate increases from 200 to 400.

However, the transaction latency increased from 5 seconds to 30 seconds, the response time increased from 10 seconds to 50 seconds, the block propagation time increased from 0.5 seconds to 10 seconds, and the consensus time increased from 1 minute to 10 minutes. These results suggest that increasing the transaction rate can improve network throughput but can also negatively impact transaction latency, block propagation time, and consensus time.

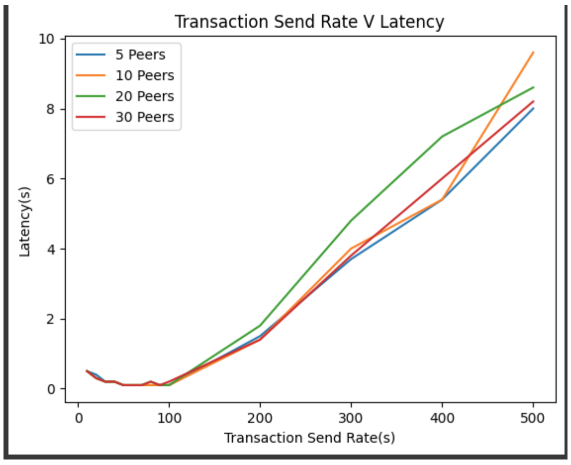


Figure 14. Transaction Send Rate vs Latency

6.3 Impact of Number of Peers

Figures 15 and 16 illustrate the impact of the number of peers on the performance of the blockchain system in terms of latency, throughput, block propagation time, and consensus time. The experiments revealed that the system's performance degrades as the number of participating nodes increases from 10 to 100.

The system was tested with different transaction rates and numbers of peers. The results showed that at 100 transactions per second (tps), the throughput decreased from 51.00 tps [15] with five peers to 34.00 tps with 100 peers (Figure 15). At 200 tps, the throughput decreased from 51.25 tps to 31.50 tps, with an increase in peers from 5 to 100. Although the system reached its peak throughput of 51.50 tps at 300 tps with five peers, the throughput decreased to 33.00 tps with 100 peers.

As shown in Figure 15, the latency increased with the number of peers. At 100 tps, the latency increased from 3 ms with five peers to 8 ms with 40 peers and remained constant at 8-9 ms for more than 40 peers. At 200 tps, the latency increased from 8 ms with five peers to 20 ms with 40 peers and remained constant at 18-21 ms for more than 40 peers. Similarly, at 300 tps, the latency increased from 13 ms with five peers to 31 ms with 40 peers and remained constant at 28-31 ms for more than 40 peers.

The block propagation time also increased with the number of peers, as shown in Figure 16. At 100 tps, the block propagation time increased from 12.1 ms with five peers

[15] all the tps reported in this subsection were scaled down by a factor 10 for plotting purposes

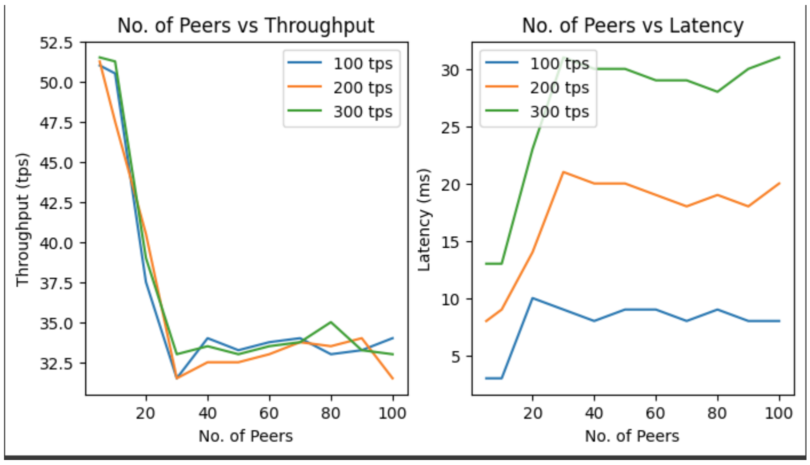


Figure 15. Number of Nodes Vs. Latency / Throughput

to 43 ms with 100 peers. At 200 tps, the block propagation time increased from 12.6 ms with five peers to 45.15 ms with 100 peers. At 300 tps, the block propagation time increased from 13.356 ms with five peers to 47.829 ms with 100 peers.

Finally, the consensus time also increased with the number of peers. At 100 tps, the consensus time increased from 0.98 s with five peers to 1.83 s with 100 peers (Figure 16). Similarly, at 200 tps, the consensus time increased from 1.06 s with five peers to 1.98 s with 100 peers. At 300 tps, the consensus time increased from 1.16 s with five peers to 2.16 s with 100 peers.

6.4 Time Taken for Sensor Data to reach MinIO

We tested different sensor send rates, ranging from 100 to 3000 data points every 5 seconds. Since the SBN and the PBN would also modify the payload before it reaches the MinIO storage server, we compared the time stamps of the retrieved payload on MinIO. Our analysis showed that for send rates below 3000 data points every 5 seconds, a sensor payload took an average of 0.024 to 0.026 seconds to pass through both blockchain networks and reach the MinIO server. The delay at the SBN was responsible for about 78

7 Answers to Research Questions

This section answers the research questions in the following ways:

What are the best security measures for edge computing environments to protect sensitive data?

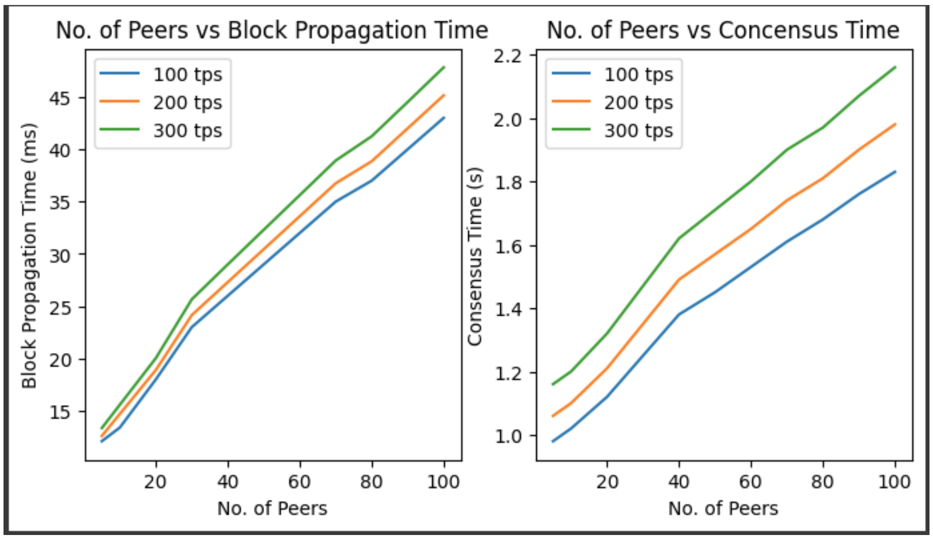


Figure 16. Number of Nodes Vs. Block Propagation/Consensus Time.

The proposed E2C-Block architecture provides an effective solution for protecting sensitive data in edge computing environments. It utilizes blockchain technology, which provides a secure and tamper-proof system for processing, transmitting, and storing sensor data. The MinIO cloud-based storage system complements the blockchain networks, providing an efficient and reliable off-chain storage solution. The architecture takes a comprehensive approach to data security, guaranteeing that the data is processed, transmitted, and stored securely and tamper-proof. This approach provides a dependable and effective method of protecting confidential information and preventing unauthorized access, ultimately enhancing the security and privacy of the system.

How can blockchain, fog, and edge computing be integrated to overcome scalability and reduce storage costs?
The proposed E2C-Block architecture integrates blockchain, fog computing, and edge computing to overcome scalability and reduce storage costs. The architecture uses two blockchain networks and an external data repository. The SBN is located in a fog computing environment close to the IoT sensors, providing a secure communication channel between IoT devices and the cloud infrastructure. The use of blockchain technology ensures data integrity and tamper-proof storage, while the external data repository provides a scalable and secure storage solution. Storing the hash of the sensor data on the PBN instead of the data itself reduces the storage footprint significantly while ensuring data integrity.

How can blockchain technology be leveraged to ensure the integrity and immutabil-

ity of data stored in fog and edge computing environments?
The proposed E2C-Block architecture leverages blockchain technology to ensure the integrity and immutability of data stored in fog and edge computing environments. The architecture uses two blockchain networks. The SBN provides a secure communication channel between IoT devices and the cloud infrastructure, and the PBN ensures data integrity and tamper-proof storage. Storing the hash of the sensor data on the PBN instead of the data itself ensures data integrity while reducing the storage footprint significantly. Moreover, the data can be retrieved from the MinIO server and compared against the hash value previously stored on the PBN to verify its validity, ensuring that the data has not been tampered with or altered.

7.1 Limitations

The proposed E2C-Block architecture provides data security and integrity benefits but has limitations that should be evaluated before using it. The architecture's evaluation was specific to a use case and may not be suitable for high data volume or real-time data processing scenarios. The use of MinIO for off-chain storage is efficient but can impact overall system performance and availability. The architecture's effectiveness against adversarial attacks and malicious actors is unknown. It can also introduce computational and storage overhead and may perform poorly in environments with limited network resources or high latency. Further research is necessary to address these limitations and validate the architecture's effectiveness in real-world scenarios. Hashing was performed on every sensor data point, which could be computationally demanding for large sensor data collections.

7.2 Future Works

- Deployment Automation: The current deployment automation of E2C-Block using Ansible and Fablo could be improved by creating more robust and configurable scripts and tools. Automating the deployment process will make it easier to set up and manage all the components of the E2C-Block environment. This will make the deployment process more efficient, especially when deploying across a distributed list of servers.

- Pluggable Consensus Algorithms: Adding support for additional consensus algorithms like Kafka would make E2C-Block more flexible and provide users with more options. This will enhance the performance of E2C-Block and make it more adaptable to different use cases. It is crucial to consider the specific requirements of each use case to determine the most suitable consensus algorithm.

- Making it Truly Distributed: To support production-level workloads, it is essential to move all the components of the Hyperledger framework of E2C-Block to separate hosts, making it truly distributed. By doing so, E2C-Block can utilize more robust consensus algorithms like Kafka. This will improve the overall performance and scalability of E2C-Block .

- Mapping More than Data Points to Hashing: The hashing process of the PBN of E2C-Block should be optimized to hash a block of sensor data instead of a single data payload. This will reduce the number of hashes required for the same number of sensor data payloads, improving the performance of E2C-Block . Additionally, adjustments should be made to how the data authenticity check is performed in the PBN when querying the MinIO Storage Server to accommodate this change.

8 Conclusion

The thesis addressed three critical research questions concerning securing edge computing environments, using blockchain for data integrity and immutability, and integrating fog computing and edge computing to improve scalability and reduce storage costs. The study yielded several significant findings. Protecting edge computing environments is crucial due to their vulnerability to attacks, which can be mitigated using the SBN in the fog computing environment. Hashing sensor data on the PBN and storing data on MinIO can reduce storage costs and enhance scalability.

The experiment demonstrated that increasing the transaction rate can improve network throughput but negatively affect other performance metrics. Similarly, increasing the number of peers can also negatively impact network performance. The study also revealed that the sensor payload takes approximately 0.024 seconds to reach the MinIO storage server. In summary, the study provides valuable insights into the effective utilization of blockchain and related technologies to enhance edge computing environments' security, scalability, and performance. Overall, E2C-Block provides an effective solution for managing and securing IoT sensor data.

Acknowledgements

I thank God for bringing me this far. I want to thank everyone who supported me while studying at the University of Tartu. I want to thank my supervisors, Dr. Chinmaya Kumar Dehury and Dr. Mubashar Iqbal, for their valuable comments and support while I worked on my thesis. I thank my wife, Ave and Barr Elizabeth, for their support.

References

[1] Seham Alnefaie, Asma Cherif, and Suhair Alshehri. A distributed fog-based access control architecture for iot. *KSII Transactions on Internet and Information Systems*, 15(12):4545 – 4566, 2021.

[2] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Communications of the ACM*, 53(4):50 – 58, 2010.

[3] Malvinder Singh Bali, Kamali Gupta, Deepika Koundal, Atef Zaguia, Shubham Mahajan, and Amit Kant Pandit. Smart architectural framework for symmetrical data offloading in iot. *Symmetry*, 13(10), 2021.

[4] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Fog computing and its role in the internet of things*, page 13 – 15, 2012.

[5] Ryan A. Cooke and Suhaib A. Fahmy. A model for distributed in-network and near-edge computing with heterogeneous hardware. *Future Generation Computer Systems*, 105:395 – 409, 2020.

[6] Chinmaya Dehury, Satish Narayana Srirama, Praveen Kumar Donta, and Schahram Dustdar. Securing clustered edge intelligence with blockchain. *IEEE Consumer Electronics Magazine*, pages 1–1, 2022.

[7] Elahe Fazeldehkordi and Tor-Morten Grønli. A survey of security architectures for edge computing-based iot. *IoT*, 3(3):332 – 365, 2022.

[8] HLF Foundation. Peers documentation. `https://hyperledger-fabric.readthedocs.io/en/release-2.5/peers/peers.html`. 22.04.2023.

[9] HLF Foundation. Smart contracts. `https://www.hyperledger.org/wp-content/uploads/2018/04/Hyperledger_Arch_WG_Paper_2_SmartContracts.pdf`, 2018. 2023-04-22.

[10] HLF Foundation. Orderers. https://hyperledger-fabric.readthedocs.io/en/release-2.5/orderer/ordering_service.html, 2021. 22.04.2023.

[11] HLF Foundation. Benchmarking hlf 2.5. https://www.hyperledger.org/blog/2023/02/16/benchmarking-hyperledger-fabric-2-5-performance, 2023. 22.04.2023.

[12] HLF Foundation. Ledger documentation. https://hyperledger-fabric.readthedocs.io/en/latest/ledger/ledger.html, 2021. 20.03.2023.

[13] Hyperledger. Hyperledger case study: Walmart. https://www.hyperledger.org/wp-content/uploads/2019/02/Hyperledger_CaseStudy_Walmart_Printable_V4.pdf, 2019.

[14] Hyperledger Fabric. Network Configuration. https://hyperledger-fabric.readthedocs.io/en/latest/network/network.html, 2021. [Accessed: April 12, 2023].

[15] N. Krishnaraj, A. Daniel, Kavita Saini, and Kiranmai Bellam. Edge/fog computing paradigm: Concept, platforms and toolchains. *Advances in Computers*, 127:413 – 436, 2022.

[16] Ruonan Li, Yang Qin, Canhui Wang, Mengya Li, and Xiaowen Chu. A blockchain-enabled framework for enhancing scalability and security in iiot. *IEEE Transactions on Industrial Informatics*, page 1–11, 2022.

[17] Bin Liu, Xiao Liang Yu, Shiping Chen, Xiwei Xu, and Liming Zhu. Blockchain based data integrity service framework for iot data. In *Blockchain Based Data Integrity Service Framework for IoT Data*, page 468 – 475, 2017.

[18] Chuanwen Luo, Liya Xu, Deying Li, and Weili Wu. Edge computing integrated with blockchain technologies. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12000 LNCS:268 – 288, 2020.

[19] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Bitcoin.org*, 2008.

[20] Mandla Ndzimakhwe, Arnesh Telukdarie, Inderasan Munien, Andre Vermeulen, Uche K. Chude-Okonkwo, and Simon P. Philbin. A framework for user-focused electronic health record system leveraging hyperledger fabric. *Information (Switzerland)*, 14(1), 2023.

[21] R3. Corda: An introduction. Whitepaper, 2016.

[22] Yongjun Ren, Yan Leng, Yaping Cheng, and Jin Wang. Secure data storage based on blockchain and coding in edge computing. *Mathematical Biosciences and Engineering*, 16(4):1874 – 1892, 2019.

[23] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637 – 646, 2016.

[24] Enchang Sun, Kang Meng, Ruizhe Yang, Yanhua Zhang, and Meng Li. Research on distributed data sharing system based on internet of things and blockchain. *Journal of Systems Science and Information*, 9(3):239 – 254, 2021.

[25] Zia Ullah, Basit Raza, Habib Shah, Shahzad Khan, and Abdul Waheed. Towards blockchain-based secure storage and trusted data sharing scheme for iot environment. *IEEE Access*, 10:36978 – 36994, 2022.

[26] University of Tartu. Ut rocket, 2018.

[27] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. An overview of blockchain technology: Architecture, consensus, and future trends. In *An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends*, page 557 – 564, 2017.

Appendix

Access to Code

The code described in this thesis can be found in the public GitHub repository.
https://github.com/chinmaya-dehury/Blockchain4E2CC.

Licence

Non-exclusive licence to reproduce thesis and make thesis public