AN END-TO-END DECENTRALIZED INTERNET OF THINGS (IOT) DATA MODEL

by

Krishnateja Vemula

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
August 2023

Table of Contents

List of Figures . . . . . **iv**

List of Abbreviations Used . . . . . **vii**

Abstract . . . . . **ix**

Acknowledgements . . . . . **xi**

Chapter 1 Introduction **1**

1.1 Motivation 3

1.2 Contribution 4

Chapter 2 Background Knowledge and Literature Review **7**

2.1 Decentralized Network 7

2.2 Blockchain Trilemma 8

2.3 Decentralized Storage 10

2.4 Decentralized Data Management 12

2.5 Integration of IOTA Technologies in IoT 13

2.6 Iota Streams 14
2.6.1 Channels Protocol 14

2.7 Swarm 17

2.8 Literature Review on decentralized data models 17

Chapter 3 Methodology **21**

3.1 Research Design 21

3.2 System Design 21
3.2.1 Architectural Design 22
3.2.2 IOTA Streams Integration 24
3.2.3 Swarm Storage Integration 27
3.2.4 Why IOTA Streams with Swarm? 27

3.3 Implementation 28
3.3.1 IOTA Streams Implementation 29

3.3.2 Swarm Storage Implementation 32

Chapter 4 Experimental Results and Evaluation 37

4.1 Evaluation 37
4.1.1 Data Latency 37
4.1.2 System Throughput 43
4.1.3 Data Integrity 51

Chapter 5 Conclusion and Future Work 55

5.0.1 Limitations and Future Work: 55
5.0.2 Conclusion 55

Bibliography 57

List of Figures

1.1 Directed Acyclic graph Tangle [1] 1

2.1 Blockchain Trilemma 8

3.1 Overall Architectural Design of an end-to-end decentralized data model 22

3.2 Sample Use case for the model proposed 23

3.3 System Architectural Diagram of an end-to-end decentralized data model 28

3.4 IoTA Streams package importing in code 29

3.5 IoTA Node Settings 29

3.6 Creating New Seed 29

3.7 Creating Author and Channel 30

3.8 Sending Announcement Link 30

3.9 Creating Subscriber 30

3.10 Received announcement and Subscription Link 30

3.11 Received Subscription link and send Keyload message 31

3.12 Synchronizing Channel State and Sending Tagged packet . . . 31

3.13 Fetching New Messages from Channel 31

3.14 Sync channel state and send multiple signed packets 31

3.15 Fetching multiple new messages from Channel 32

3.16 Bee Storage package importing in code 32

3.17 Bee node instance 32

3.18 Bee Storage Adapter Module 33

3.19 Bee Integration with IoTA Streams 33

3.20 Public Single Branch Single Publisher Communication 34

3.21 Private Single Branch Single Publisher Communication 34

3.22 Public Single Depth Single Publisher Communication 34
3.23 Private Single Branch Single Publisher Communication 35
3.24 Private Multiple Branch Single Publisher per branch communication 35
3.25 Private Multiple Branch Multiple Publisher per branch communication 36
3.26 Mixed Multi Branch Single Publisher communication 36

4.1 No of Packets vs Execution Time 38
4.2 Distribution of Packets for each Time across Packets 39
4.3 Outliers showing the Average number of Packets transmitted per time interval 39
4.4 No.of Packets vs Time 40
4.5 Distribution of packets for each Time 41
4.6 Explanation of Packets by Time 42
4.7 Time for Complete execution and Packet execution 43
4.8 CPU Usage vs Packets 44
4.9 Explanation of CPU Usage by Packets 44
4.10 Heatmap of CPU Usage against packets 46
4.11 CPU Usage in ms against Number of Packets 47
4.12 CPU User Usage in ms against Number of Packets 48
4.13 CPU System Usage in ms against Number of Packets 48
4.14 CPU System and User Usage in ms against Number of Packets 49
4.15 Average packets against CPU Usage in percent, packets, CPU user, and system usage 50
4.16 A Hash of the payload on the tangle 52
4.17 Data confirmation on IOTA Tangle 52
4.18 Hash to text Conversion of payload 52
4.19 Validation code snippet 53

4.20 Modify the data payload 53
4.21 Extra Field detection 54

List of Abbreviations Used

Acronyms

API Application Programming Interface

DAG Directed Acyclic Graph

DHT Distributed Hash Table

DID Decentralized Identifiers

DLT Distributed Ledger Technology

DPos Delegated Proof of Stake

HTTP Hypertext Transfer Protocol

IOTA An open source of distributed ledger technology

IoT Internet of Things

IPFS Inter Planetary File System

IP Internet Protocol

LPWAN Low Power Wide Area Network

M2M Machine to Machine

MAM Masked Authenticated Messaging

MQTT Message Queuing Telemetry Transport

PBFT Practical Byzantine Fault Tolerance

PoA Proof of Authority

POST Power-On Self-Test

PoS Proof of Stake

PoW Proof Of Work

RSA Rivest, Shamir, Adleman public key encryption Algorithm

sha256 Secure Hash Algorithm 256-bit

TPS Transactions per second

Abstract

In today's data-driven society, the demand for secure, efficient, and decentralized data storage and communication solutions has become increasingly critical. This thesis explores the design and implementation of an end-to-end decentralized data model that leverages IOTA Streams and Swarm Storage technologies.

The research begins by examining the limitations and challenges associated with centralized data models, such as single points of failure, data breaches, and high maintenance costs. To overcome these challenges, a decentralized approach is proposed, which combines the strengths of IOTA Streams and Swarm Storage.

IOTA Streams is a messaging and data transmission protocol that provides a secure and tamper-proof communication layer. It allows for end-to-end encryption, data integrity verification, and fine-grained access control. By utilizing IOTA Streams, the proposed data model ensures the confidentiality and integrity of data throughout its lifecycle.

Swarm Storage, on the other hand, offers a decentralized and fault-tolerant storage infrastructure. It employs a distributed hash table (DHT) network to store and retrieve data, enabling data redundancy, availability, and scalability. The integration of Swarm Storage within the data model enhances data persistence and accessibility in a decentralized manner.

The thesis presents the design and implementation details of the end-to-end decentralized data model. It covers aspects such as data ingestion, transmission, storage, retrieval, and access control. The model incorporates cryptographic techniques, consensus mechanisms, and decentralized identifiers (DIDs) to establish a robust and secure data ecosystem.

To evaluate the performance and effectiveness of the proposed data model, a series of experiments and simulations are conducted. The experiments assess factors such as data transfer speed, storage efficiency, fault tolerance, and resilience against attacks. The results validate the feasibility and advantages of the end-to-end decentralized data model based on IOTA Streams and Swarm Storage.

In conclusion, this thesis demonstrates the potential of combining IOTA Streams and Swarm Storage to achieve an end-to-end decentralized data model. The proposed solution addresses the limitations of centralized data models and provides a secure, efficient, and scalable alternative for storing and transmitting data in decentralized environments. The findings of this research contribute to the field of decentralized systems and can guide the development of future decentralized data solutions.

Acknowledgements

I am deeply grateful to my supervisor, **Dr. Srinivas Sampalli**, who patiently guided and mentored me in this project.

I am very grateful to **Dr. Marzia Zaman** for her guidance and feedback throughout the research.

I am thankful for the support that was provided by **NSERC and Cistel Technologies** through Collaborative Research and Development grants.

Chapter 1

Introduction

In recent years, the proliferation of the Internet of Things (IoT) [2] devices has generated an unprecedented amount of data that needs to be stored, transmitted, and managed securely. Traditional centralized data models [3], characterized by their reliance on single points of failure and vulnerability to data breaches, are ill-suited to meet the demands of this data-intensive landscape. To address these challenges, decentralized data models have emerged as promising alternatives, offering increased security, efficiency, and scalability [4]. In this thesis, we explore the design and implementation of an end-to-end decentralized data model that leverages IOTA Tangle, IOTA Streams, and Swarm Storage, with a focus on IoT devices and data transfer.

The IOTA Tangle is a distributed ledger technology designed specifically for the IoT ecosystem. Unlike traditional blockchain architectures, the IOTA Tangle employs a directed acyclic graph (DAG) structure, where each transaction confirms two previous transactions, forming a web of interconnected transactions as shown in Figure 1.1. This unique structure enables high scalability and eliminates the need for miners, resulting in feeless and lightweight transactions [5].

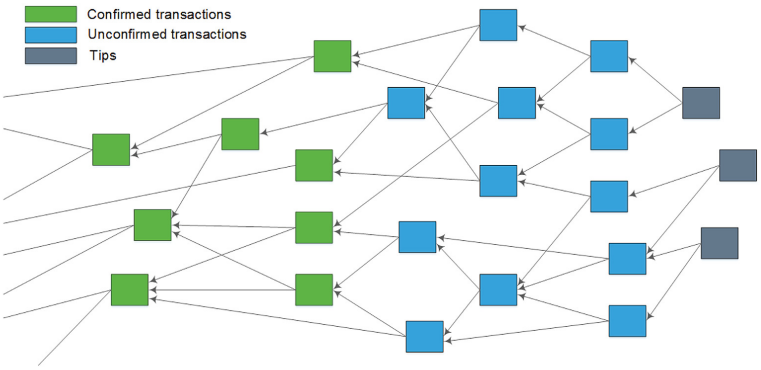


Figure 1.1: Directed Acyclic graph Tangle [1]

By integrating IOTA Tangle into our decentralized data model, we establish a

reliable and tamper-proof transaction layer for IoT devices.

Building upon the IOTA Tangle, IOTA Streams provides a secure and decentralized messaging and data transmission protocol. With IOTA Streams, data can be securely shared between IoT devices, ensuring confidentiality, integrity, and fine-grained access control. It enables end-to-end encryption, data authentication, and granular permissions, allowing data owners to maintain control over their information [6]. By incorporating IOTA Streams into our data model, we ensure that data is securely transmitted and protected throughout its lifecycle.

While IOTA Streams addresses the communication aspect of our decentralized data model, Swarm Storage serves as the decentralized storage infrastructure. Swarm Storage utilizes a distributed hash table (DHT) network, where data is fragmented and distributed across multiple nodes. This approach ensures data redundancy, availability, and scalability, eliminating the reliance on centralized servers. By leveraging Swarm Storage, our data model achieves fault tolerance, improved data persistence, and accessibility in a decentralized manner [7].

The convergence of IoT devices and decentralized data models presents unique challenges in achieving secure and efficient data transfer. IoT devices are often resource-constrained, operating with limited processing power and bandwidth. Additionally, ensuring data privacy and security in a decentralized environment poses significant technical hurdles [8]. Thus, this thesis aims to address these challenges by proposing an end-to-end decentralized data model that optimizes data transfer for IoT devices while maintaining robust security and privacy mechanisms.

In summary, this thesis explores the development of an end-to-end decentralized data model, specifically tailored to the requirements of IoT devices and data transfer. By incorporating the IOTA Tangle, IOTA Streams, and Swarm Storage, we aim to establish a secure, scalable, and efficient solution for managing IoT data in a decentralized manner. The subsequent chapters of this thesis delve into the design, implementation, and evaluation of our proposed data model, shedding light on its performance, advantages, and potential impact on the field of decentralized systems and IoT data management.

1.1 Motivation

The motivation behind this research stems from the pressing need for secure, efficient, and decentralized data management solutions in the context of IoT devices and data transfer. As the number of IoT devices continues to soar and the volume of generated data exponentially increases, traditional centralized data models struggle to cope with the associated challenges and limitations [9]. This motivates the exploration of decentralized alternatives that can address the shortcomings of centralized approaches and provide a more robust and scalable framework for IoT data management.

One key motivation is the inherent vulnerabilities of centralized data models. Centralized systems are susceptible to single points of failure, making them highly fragile and prone to service disruptions. Moreover, the centralized storage of sensitive data raises concerns about data privacy and security, as unauthorized access or data breaches can have severe consequences [3]. These limitations undermine the trustworthiness and reliability of centralized infrastructures, prompting the need for decentralized data models that distribute data storage and management across a network of participants.

Furthermore, the inefficiency of data transfer in centralized models poses significant challenges in the IoT domain. IoT devices often operate with limited resources, including processing power, bandwidth, and energy. The centralized approach requires data to be transmitted to and from a central server, resulting in increased latency, network congestion, and higher energy consumption. These inefficiencies hinder real-time decision-making, delay critical actions, and hinder the scalability of IoT deployments. Therefore, there is a need for decentralized data models that optimize data transfer, minimize latency, and reduce the burden on resource-constrained IoT devices.

The emergence of technologies such as IOTA Tangle, IOTA Streams, and Swarm Storage provides a unique opportunity to address the aforementioned challenges. IOTA Tangle offers a decentralized and scalable distributed ledger specifically designed for the IoT ecosystem. Its feeless transactions, lightweight nature, and tamper-proof characteristics make it an ideal foundation for decentralized data models. Additionally, IOTA Streams provides a secure and granular data transmission protocol,

ensuring data confidentiality, integrity, and access control. Combined with Swarm Storage, which offers fault-tolerant and scalable decentralized storage, these technologies offer a comprehensive framework for developing an end-to-end decentralized data model.

The potential benefits of such a decentralized data model are substantial. It can enhance data privacy, security, and ownership, empowering individuals and organizations to retain control over their data in a trustless environment. The decentralized nature of the model improves resilience against attacks, as there is no single point of vulnerability. Furthermore, by optimizing data transfer for IoT devices, the proposed model can improve overall system performance, reduce latency, and enable real-time decision-making, ultimately unlocking the full potential of the IoT.

In conclusion, the motivation behind this research lies in the need to address the limitations of centralized data models in the context of IoT devices and data transfer. By leveraging technologies such as IOTA Tangle, IOTA Streams, and Swarm Storage, we aim to develop an end-to-end decentralized data model that provides secure, efficient, and scalable solutions for IoT data management. The outcomes of this research have the potential to significantly impact the field of decentralized systems, IoT deployments, and data management practices, paving the way for a more trustworthy and resilient digital future.

1.2 Contribution

This thesis makes several contributions to the field of decentralized data management, specifically focusing on IoT devices and data transfer. The key contributions of this research are as follows:

1. **Development of an End-to-End Decentralized Data Model:** The thesis proposes a comprehensive end-to-end decentralized data model that addresses the limitations of centralized approaches in the context of IoT devices and data transfer. By integrating IOTA Tangle, IOTA Streams, and Swarm Storage, the model establishes a secure, efficient, and scalable framework for managing IoT data in a decentralized manner.

2. **Integration of IOTA Tangle for Transaction Layer:** The research contributes to the integration of IOTA Tangle, a distributed ledger technology, as the transaction layer of the decentralized data model. This integration leverages the unique properties of IOTA Tangle, such as its lightweight and feeless nature, to provide a reliable and tamper-proof infrastructure for IoT device transactions.

3. **Utilization of IOTA Streams for Secure Data Transmission:** The thesis explores the utilization of IOTA Streams, a secure messaging and data transmission protocol, within the decentralized data model. By incorporating IOTA Streams, the model ensures end-to-end encryption, data integrity verification, and fine-grained access control, thereby addressing data privacy and security concerns in a decentralized environment.

4. **Incorporation of Swarm Storage for Decentralized Data Storage:** The research contributes to the incorporation of Swarm Storage, a decentralized storage infrastructure, into the proposed data model. By utilizing Swarm Storage's distributed hash table (DHT) network, the model achieves data redundancy, availability, and scalability, eliminating the reliance on centralized servers and enhancing fault tolerance and data persistence.

5. **Design and Implementation of Data Ingestion and Retrieval Mechanisms:** The thesis presents the design and implementation details of data ingestion and retrieval mechanisms within the decentralized data model. These mechanisms facilitate seamless and efficient data transfer between IoT devices and the decentralized storage infrastructure, optimizing resource utilization and reducing latency.

6. **Evaluation of Performance and Effectiveness:** The research conducts a series of experiments and simulations to evaluate the performance and effectiveness of the proposed data model. The evaluations assess factors such as data transfer speed, storage efficiency, fault tolerance, and resilience against attacks, providing insights into the advantages and limitations of the decentralized data model in practical scenarios.

7. **Contribution to Decentralized Systems and IoT Data Management:** The findings of this research contribute to the broader field of decentralized systems and IoT data management. The proposed data model presents a novel approach for securely and efficiently managing IoT data in a decentralized manner, addressing the challenges associated with centralized models. The research outcomes can guide the development of future decentralized data solutions, improving data privacy, security, and ownership in IoT ecosystems.

In conclusion, this thesis makes significant contributions by proposing an end-to-end decentralized data model that leverages IOTA Tangle, IOTA Streams, and Swarm Storage for IoT devices and data transfer. The integration of these technologies, along with the design and implementation of data ingestion and retrieval mechanisms, offers a secure, efficient, and scalable solution for decentralized data management. The research findings contribute to the advancement of decentralized systems and IoT data management practices, paving the way for a more trustworthy and resilient data ecosystem.

Chapter 2

Background Knowledge and Literature Review

2.1 Decentralized Network

Decentralized networks have gained significant attention in recent years due to the growing concerns over privacy, data ownership, and control in centralized social networks. In a decentralized network, the system operates on independently run servers rather than a centralized server owned by a single business. This approach promotes user anatomy, and data privacy, and is resilient against censorship and a single point of failure.

1. **Decentralized Network Architecture:** Research has explored various decentralized network architectures beyond blockchain, such as Directed Acyclic Graphs (DAGs), Tangle, and Holochain. DAG-based architectures, exemplified by IOTA's Tangle, offer scalability and transaction parallelism, making them suitable for IoT and microtransactions [10]. Holochain, on the other hand, provides a scalable framework for decentralized applications with agent-centric data models [11].

2. **Consensus mechanism:** Extensive research has been conducted on consensus mechanisms to address the limitations of traditional PoW and PoS [12, 13]. Alternative approaches, such as Delegated Proof-of-Stake (DPoS) [14, 15], Practical Byzantine Fault Tolerance (PBFT)[16], Directed Acyclic Graph(DAG) [12], and Proof-of-Authority (PoA)[17], have been proposed to improve scalability, energy efficiency, and transaction throughput in decentralized networks.

3. **Scalability and Performance:** Scalability and performance are still important issues in blockchain networks; substantial research has been conducted comparing various types of blockchain networks in terms of scalability problems and performance features[18, 19]. Because of the vast number of participating

nodes, public blockchains have scalability constraints[20, 19], but private[21] and permissioned blockchains[22] have more scalability possibilities owing to regulated network conditions. To achieve optimal scalability and performance in blockchain networks, however, consensus algorithms, network architecture, governance frameworks, and the deployment of appropriate scaling solutions must all be carefully considered[18, 20, 19, 22, 21]. More research and development are required to overcome scalability issues and increase the performance of blockchain networks across various deployment options.

2.2 Blockchain Trilemma

The "blockchain trilemma" is a concept that highlights the trade-offs between three critical properties in blockchain networks: decentralization, scalability, and security. The trilemma suggests that it is challenging to achieve high levels of all three properties simultaneously. As a result, most blockchain networks prioritize two of the three properties at the expense of the third as shown in Figure 2.1 [23]. For example:

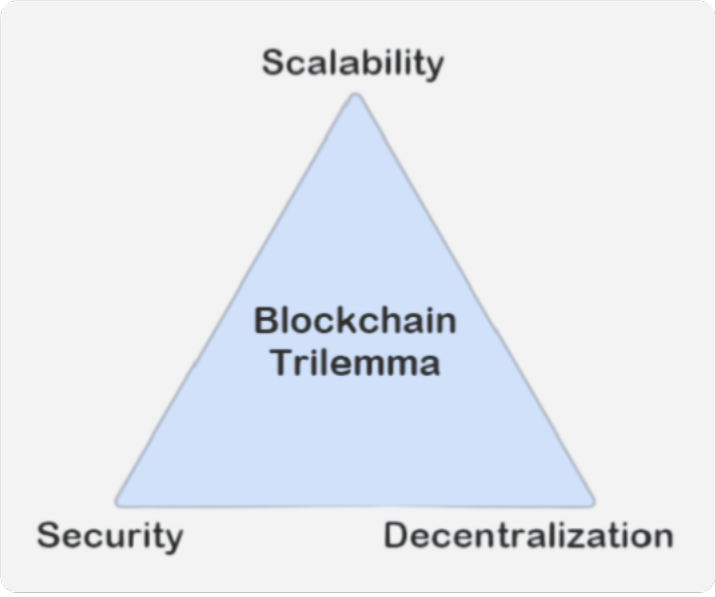


Figure 2.1: Blockchain Trilemma

1. **Decentralization**: Refers to the distribution of control and decision-making power across multiple nodes in the network. A highly decentralized blockchain involves many participants (nodes) validating transactions, which enhances censorship resistance and resilience against attacks. However, achieving high decentralization can lead to increased network latency and reduced scalability.

2. **Scalability**: Refers to the ability of a blockchain network to process a large number of transactions per second (TPS) efficiently. High scalability allows the network to handle a growing number of users and transactions without significant performance degradation. However, achieving high scalability can often require compromising on decentralization or security.

3. **Security**: Refers to the resistance against attacks and malicious behavior on the blockchain network. Robust security ensures that the transactions are immutable and tamper-proof. However, enhancing security can sometimes lead to reduced scalability and require additional computational resources.

IOTA and Swarm Storage combined aim to address the blockchain trilemma by offering a decentralized, scalable, and secure solution for data management and communication.

IOTA Tangle and Decentralization: IOTA Tangle, the underlying architecture of IOTA, takes a different approach from traditional blockchain networks. It utilizes a Directed Acyclic Graph (DAG) structure where each transaction confirms two previous transactions, creating a web of interconnected transactions as shown in Figure 1.1. This structure promotes a high level of decentralization because each participant can directly validate transactions without relying on specialized miners. By removing the need for miners and fees, IOTA Tangle achieves a high degree of decentralization while remaining lightweight and efficient.

Swarm Storage and Scalability: Swarm storage complements the IOTA Tangle's decentralization by providing a scalable and distributed storage solution. Swarm utilizes a Distributed Hash Table (DHT) network, where data is divided into smaller chunks and distributed across multiple nodes in the network. This sharding approach enables efficient data retrieval and storage, allowing Swarm to scale with the size of

the network. As more nodes join the network, the storage capacity increases, ensuring high scalability without compromising on decentralization.

End-to-End Encryption with IOTA Streams for Security: Data privacy and security are critical in decentralized data models. IOTA Streams, a protocol built on top of IOTA Tangle, offers end-to-end encryption and secure data communication. It allows data publishers to encrypt messages and share them securely with authorized subscribers. The integration of IOTA Streams with IOTA Tangle and Swarm ensures that data remains encrypted, authenticated, and tamper-proof throughout its lifecycle, providing robust security in a decentralized environment.

Conclusion: By combining IOTA Tangle's decentralized architecture, Swarm Storage's scalable data management, and IOTA Streams secure communication, IOTA and Swarm present a compelling solution to the blockchain trilemma. This end-to-end decentralized data model offers high levels of decentralization, scalability, and security, making it suitable for various applications, including Internet of Things (IoT) devices, where data privacy, efficiency, and resilience are of utmost importance. The convergence of these technologies paves the way for a new era of decentralized systems that can efficiently handle massive amounts of data while preserving the principles of trust and security in a decentralized network.

2.3 Decentralized Storage

Decentralized storage is a potential approach for addressing the constraints and risks of centralized storage structures. Significant research has been undertaken in different decentralized storage technologies, such as Storj, IPFS, Swarm, Sia, and Filecoin[15]. To provide secure, scalable, and robust storage solutions, these platforms make use of blockchain technology, peer-to-peer networks, and unique storage technologies. Researchers are working on solving difficulties in decentralized storage systems such as performance optimization, data availability, and incentive mechanisms. Decentralized storage improvements can pave the way for more secure and dependable storage infrastructure in various sectors, including cloud storage, content distribution, and decentralized apps[24].

1. **Storj**: According to (Wilkinson and Lowder, 2014)[25], it is time for cloud storage solutions to evolve into real clouds, and blockchain enables this. Storj

leverages the Ethereum blockchain to store metadata for each block, allowing users to access their content in its entirety whenever required. Metadisk monitors the network regularly to ensure that the files stored are accessible and unaltered. If a node or audit is not accessible, a new node is used to save the file. All files in Storj are broken into small portions before being transferred across the network. This conceals crucial data by combining it with irrelevant information.[24, 25]

The author of [24] presented Storj, a decentralized cloud storage platform based on blockchain technology and peer-to-peer networks. The research assessed Storj's scalability, performance, and security, emphasizing its ability to deliver safe and dependable storage at a reasonable cost when compared to established cloud storage providers.

2. **IPFS**: The InterPlanetary File System (IPFS), a technology that allows for decentralized and distributed file storage, was explained by the author [26]. The study looked into IPFS's architecture and functionality, emphasizing its potential for data deduplication, content-addressable storage, and peer-to-peer file sharing[27]. The study also evaluated IPFS's performance and scalability in several use scenarios[26].

3. **Sia**: Author[24, 28] investigated Sia, a blockchain-based decentralized storage platform. similar to Storj, where peers rent out their hardware capacity. Before transmitting data to the network, it must be encrypted and digitally signed, and a contract must be established between the client and hosts[28]. The research analyzed Sia's features, security measures, and storage economics, focusing on its potential for cost-effective and dependable data storage. In addition, the study assessed Sia's performance and scalability in terms of file retrieval time and network throughput.

4. **Filecoin**: The author examined Filecoin, a decentralized storage network, in comparison to various storage systems[24]. FileCoin is another example of a decentralized storage network. It works on top of IPFS and leverages Proof-of-Spacetime and Proof-of-Replication to ensure that the client's data is secure over time[29]. Filecoin's consensus process, data replication mechanisms, and

economic incentives were all explored in the study. The study emphasized Filecoin's potential for scalable and secure storage, as well as its distinct market-driven approach to incentivizing storage providers[24].

2.4 Decentralized Data Management

Blockchain technology, which was originally designed for cryptocurrencies such as Bitcoin, has emerged as a promising alternative for decentralized data management. It provides a safe, transparent, and tamper-proof means for storing and exchanging data over a network of nodes. Blockchain technology can assist address difficulties connected to data security, privacy, and interoperability in the context of data management, making it particularly useful for industries such as healthcare, supply chain management, and finance.

1. **Blockchain Technology in Healthcare - A Systematic Review:** This systematic review investigates ongoing research on the use of blockchain technology in healthcare. The authors highlight many blockchain use cases in healthcare, including electronic health records, medical supply chain management, and clinical trials. They do, however, emphasize the need for more study to better comprehend, describe, and assess the usefulness of blockchain in healthcare[30].

2. **Decentralized Clinical Trials:** The Medical Product Development of the Future. The potential of blockchain technology in decentralized clinical trials is discussed in this article. It emphasizes the advantages of implementing blockchain for clinical trial data administration, such as increased data security, transparency, and traceability. The authors also underline the importance of more studies to address the challenges and limitations of using blockchain in clinical trials[31].

3. **A Blockchain-Enabled Medical Supply Chain:** This study suggests a medical supply chain system based on blockchain technology. The authors want to fill identified gaps in current evidence of blockchain deployment in medical supply chains by developing a robust, secure, and transparent method for handling medicinal goods[31, 32].

4. **Decentralized Data Management in Healthcare Using Blockchain Technology:** This article explores the possible benefits of employing blockchain technology in healthcare for decentralized data management. It goes through how blockchain may increase data accuracy, security, and interoperability while also lowering the expenses associated with traditional data management solutions. The authors also emphasize the importance of more study and development to fully realize the promise of blockchain in healthcare[30].

5. **Blockchain for Decentralized Data Management: Opportunities and Challenges:** This research paper examines the advantages and disadvantages of utilizing blockchain technology for decentralized data management. It emphasizes the benefits of blockchain, such as increased security, transparency, and traceability, while also addressing the technological and legislative hurdles that must be solved to fully use blockchain's promise in many industries[33].

2.5 Integration of IOTA Technologies in IoT

The integration of IOTA technologies into the IoT (Internet of Things) has the potential to change how devices connect and transact with one another completely. IOTA is a distributed ledger system (DLT) developed primarily for Internet of Things (IoT) environments. It employs a unique data structure, the Tangle, a directed acyclic graph (DAG) that allows for feeless and scalable transactions[34].

Here are some ways IOTA technologies can be integrated into IoT:

1. **Secure data transfer:** IOTA's Tangle can be used to securely transfer data between IoT devices without the need for a centralized authority. This ensures data integrity and prevents tampering[34, 35].

2. **Micropayments:** IOTA enables feeless transactions, making it suitable for micropayments between IoT devices. This can be used to create new business models, such as pay-per-use services or incentivizing data sharing between devices[34].

3. **Decentralized identity:** IOTA's Digital Identity framework can be used to

create decentralized identities for IoT devices, ensuring secure and private communication between them[35].

4. **Data marketplace:** IOTA can be used to create a decentralized data marketplace where IoT devices can sell their data to interested parties. This can help monetize data generated by IoT devices and encourage data sharing[34, 36].

5. **Smart contracts:** IOTA is working on implementing smart contracts on its platform, which can be used to automate processes and agreements between IoT devices[34, 36].

6. **Scalability:** IOTA's Tangle is designed to be highly scalable, making it suitable for large-scale IoT networks with millions of devices[34, 35].

7. **Energy efficiency:** IOTA's consensus mechanism, which is based on the Tangle, is more energy-efficient than traditional blockchain-based systems, making it more suitable for IoT devices with limited power resources[34, 37].

Overall, integrating IOTA technologies into IoT can lead to more secure, efficient, and scalable networks, enabling new business models and use cases.

2.6 Iota Streams

The IOTA Streams framework is designed to provide a secure message verification and protection protocol for data transmission over a transport layer.

The Channels protocol was developed to replace the previously used MAM library for data transmission using the Tangle as the primary transport mechanism. The channels themselves may be configured in a variety of ways, with any arbitrary combination of publishers and subscribers (although each channel can only be hosted by a single author instance)[38].

2.6.1 Channels Protocol

The Channels protocol offers the high-level API tools required for authors and subscribers to be created and communicate with the Tangle[38].

Authors

A channel author is responsible for the creation of a new channel as well as the configuration of the channel's intended structure (i.e. single branch versus multi-branch). A channel author will be able to restrict access to branches within a channel structure, as well as accept and manage user subscription messages[38].

Subscribers

A channel subscriber is an individual who is not the author of a channel. A subscriber can be created without author authorization, but to write to a branch or process any private streams, they must subscribe to the channel and have the author's approval, and process that subscription. A subscriber may also connect with a stream without completing a subscription process by using pre-shared keys instead of a subscription[38].

Branching

Branches are defined as successive groupings of messages connected to the announcement message. For public and private streaming, these branches will normally be created using either a signed packet message or a keyload message. A channel can take one of two forms:

1. **Single branch:** a message sequence that is linear (akin to a MAM stream), with each message connected to the one before it.
2. **Multi-branch:** a message sequencing that does not reliant on sequential message linking.

When creating a channel, the author will choose whether to utilize single-branching or multi-branching; this will instruct the Streams instance on how to handle sequencing. Subscribers will also be notified when they process the announcement message, ensuring that their instances are in the correct sequencing order[38].

Keyloads

A keyload message is a control and access restriction message that allows the author to state who should be allowed to decode any messages attached after it.

When creating a keyload message, there are two approaches for specifying access:

1. **Subscriber Public Keys:** Public keys are masked and sent to the author to be kept on their instance throughout the processing of subscription messages. By providing that public key in the keyload message, the author may then signify which of these users will be allowed to view subsequent messages.

2. **Pre-Shared Keys:** A predefined key is distributed to end users through means other than the subscription procedure described above. These keys can be used to provide or restrict access to a stream without the requirement for a subscription[38].

Sequencing

Sequencing is a technique integrated into streams that allows message IDs to be created in sequence regardless of the structure of the channel. Messages are identified by an indexation position within the Tangle and are generated using a combination of Application instance (channel identifier), Public key of the publisher, Previous message-id (the message is linked to), Branch number (identifier for the specific branch), and Sequencing number (the publisher's sequencing position).

As messages are posted to and received from the channel, the message identification, branch, and sequencing numbers for each publishing party are updated in a local state for user implementation. To stay in sync, user implementations can deduce and search for the next message in the series.

1. **Single Branch Sequencing**: A single-branch implementation updates each user's sequencing state to the same state and increments the sequencing number by one. To avoid out-of-sync parties, it is recommended that a single branch be utilized with just one publisher.

2. **Multi-Branch Sequencing:** In a multi-branch system, a sequencing message is transmitted in unison with every data message to allow users to determine the right message-id of a sequenced message[38].

2.7 Swarm

Swarm is an Ethereum-based decentralized storage and content delivery network. It is considered as object storage system. It aims to build a censorship-resistant, fault-tolerant, and self-sustaining infrastructure for peer-to-peer content hosting and distribution[39]. Here are some important factors to consider while researching Swarm storage:

1. **Swarm Architecture:** Swarm uses a distributed network of nodes known as "swarm nodes" to store and share data. It employs a novel method known as "content-addressable chunking," which divides data into smaller pieces and provides each chunk a unique identity (content address). After that, the pieces are saved and spread over the network[40, 41].

2. **Chunk Retrieval and Incentives:** To incentivize nodes to store and distribute content, Swarm employs a retrieval market model. Nodes are compensated with digital currency for supplying chunks to consumers on demand. The retrieval algorithm is based on proximity, which means that nodes closest to the requesting node are preferred for content delivery[39, 42].

3. **Redundancy and Fault Tolerance:** Swarm enables data redundancy by replicating chunks across several nodes. This redundancy enables fault tolerance, ensuring that data remains accessible even if some nodes fail or go down. It also enables efficient retrieval via parallelization and load balancing[39, 43].

4. **Security and Privacy:** Swarm employs encryption methods to ensure the integrity and confidentiality of data. Because chunks are encrypted and spread over numerous nodes, malicious actors find it difficult to compromise the data. Swarm also connects with the Ethereum blockchain, utilizing its security features for authentication and access control[39, 43].

2.8 Literature Review on decentralized data models

Decentralized data models have emerged as a prominent topic of research because of their potential to improve data security, privacy, and control. They are

distinguished by the absence of a centralized authority, enabling data to be stored, processed, and managed across various network nodes[44].

Naz, M et al proposed a model using the Ethereum network and IPFS that achieves security and access control by executing the access roles written in the smart contract by the owner. Users are first authenticated through RSA signatures, then submit the requested amount as a price for digital content, and are given incentives for registering reviews about data[45].

In decentralized storage systems, end-to-end encryption is employed to remove the risk of data loss associated with centralized data control. Storage providers must demonstrate that they have maintained unmodified files in this network for this period of time.

The survey paper focuses on providing an overview of blockchain-based storage systems, comparing them to cloud-based storage networks, and evaluating the benefits and drawbacks of blockchain-based storage. The authors surveyed all literature on storage in which blockchain was utilized to overcome traditional difficulties. The results and topics addressed have been compiled and tallied. Exploratory research was carried out to evaluate the cutting-edge deployment of blockchain technology in decentralized storage and concluded that Blockchain-based storage solutions address various problems of traditional storage systems while also providing data privacy and security. However, because of scalability, data analysis, and access difficulties, blockchain-based storage is still in its early stages[46].

Exploratory technology study into IOTA sensor integration, utilizing IOTA as the data layer, and produced a storage/visualization application. Sensors cannot directly interface with the DLT since the IOTA protocol uses Proof of Work to send and disseminate data over the network. We explored a methodology for sending sensor data to the IOTA/Tangle network. A web application framework (Node.js Express) running on a certain IP and port retrieves the data. When employing HTTP, MQTT, and LPWAN, the server will listen for POST requests to an endpoint, get the information, and then disseminate it on the Tangle using MAM. To better control transaction issuance and resource use, authors examined a web application that collects data from the XDK110 sensors and runs a MAM client to generate IOTA bundles and broadcast them to the IOTA full node and suggested using the IOTA

network as a distributed protocol for sensor data and visualizing all sensor data in real-time. Anyone interested in the data may immediately check its integrity in the Tangle. Because information could be entered into the distributed ledger without fees and with low latency, the IOTA protocol proved to be promising as an IoT data channel. IOTA can be used to create a decentralized data marketplace where decentralized data can be exchanged for the IOTA value token. Although distributed ledger technology has grown in popularity in recent years, typical blockchain systems have limitations and as a solution, researchers demonstrated that a DAG-based DLT, such as IOTA, has no such constraints[35].

IoT streaming devices create massive volumes of data, which are stored, processed, analyzed for value generation, and accessed via centralized systems, technologies, platforms, and services. The authors of this study developed a blockchain-based solution for resource-constrained IoT streaming devices that ensures data privacy and secrecy via a proxy re-encryption network. They demonstrated system schematics, eleven algorithms, detailed implementation details, as well as security analysis. To give users access to encrypted IoT data chunks, they deployed a proxy re-encryption network and allowed multiple readings inside the same acceptance request[47].

One of the most popular implementations of this decentralized technology is decentralized storage networks. Clients can safely send files across a fully decentralized network, eliminating the danger of data failures caused by centralized controls. This study provides an in-depth examination of blockchain-based storage systems, compares them to cloud-based storage networks, and explores future research options, and concluded that Blockchain is a revolutionary technology that has the potential to alter many industries. However, constraints and concerns with blockchain-based storage systems, such as scalability issues, data analysis, and accessibility, remain contested; it is thought that blockchain-based storage is still in the process of maturing[43].

Along with IoT, blockchain is revolutionizing the Internet by allowing the Trustless, Distributed, and Secure exchange of all value. The proposed approach stores the data it gathers by issuing transactions in IOTA's ledger, known as Tangle. A gossip mechanism is used to broadcast these transactions over the IOTA network. The data kept in the Tangle must be available only to those with the necessary authorization so Masked Authenticated Messaging(MAM) a data communication protocol is used.

In this paper, the authors present a distributed sensor node system that exchanges data in an M2M way using the IOTA protocol, an innovative distributed ledger technology, and establishes a data monetization economy paradigm. In conclusion, the study proposes a distributed sensor node system that securely gathers, stores, and analyses field data using IOTA protocol characteristics. It is intended to be resilient, flexible, and adaptable to any application domain[48].

Chapter 3

Methodology

This chapter outlines the research methodology employed in this study. This research aims to design and implement an end-to-end decentralized data model using Iota Streams and Swarm storage. The methodology is designed to answer the research questions by integrating both theoretical and practical approaches.

3.1 Research Design

The research design for this study is a combination of exploratory and experimental research. The exploratory aspect involves a comprehensive literature review on decentralized data models, Iota Streams, and Swarm storage. The experimental aspect involves the design, implementation, and testing of the proposed model.

Extensive research has been conducted on existing literature, including books, peer-reviewed articles, white papers, and online resources, to gain a comprehensive understanding of the current state of decentralized data models, Iota Streams, and Swarm storage.

The practical implementation part of the study will involve the development of a prototype model using Iota Streams and Swarm storage. This will provide first-hand data transfer in the proposed model.

3.2 System Design

As we mentioned in the previous section, our research aims to establish an end-to-end decentralized data model for IoT devices. The model aims to achieve secure, efficient, and scalable data management in a decentralized network. The Research follows a mixed-method approach, combining both the design and implementation of the data model.

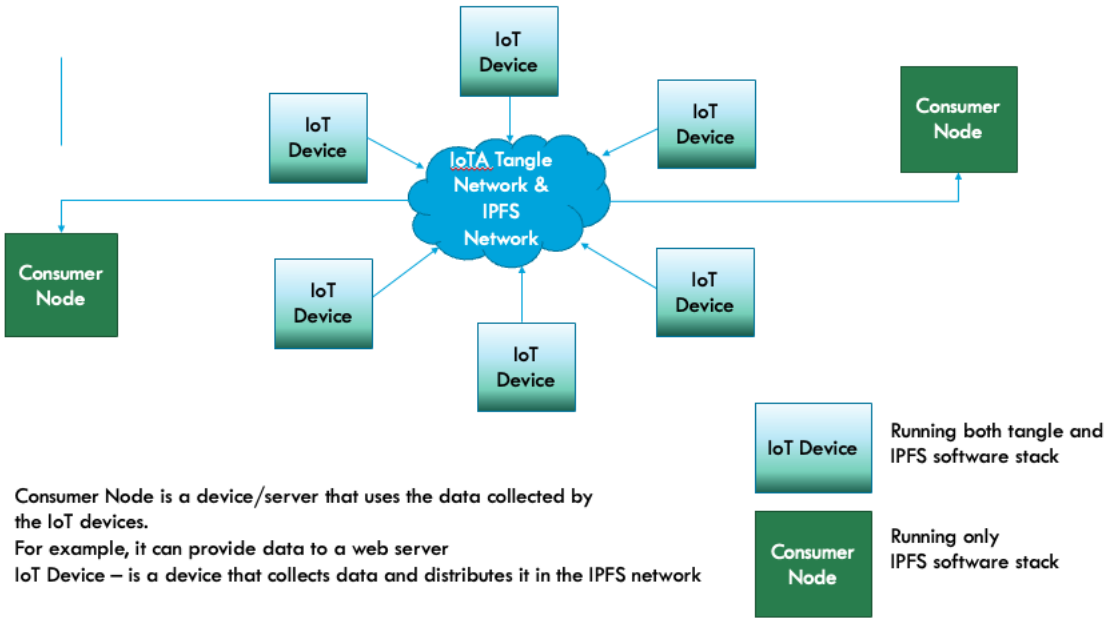


Figure 3.1: Overall Architectural Design of an end-to-end decentralized data model

3.2.1 Architectural Design

Figure 3.1 provides a comprehensive visual representation of our model, structured around four central components: the IoT Device, Consumer Node, IoTA Tangle and Streams module, and Swarm Storage module. These components, with their respective sub-modules and functionalities, form an integrated architecture ensuring efficient data transfer, storage, and retrieval in a decentralized environment.

1. **IoT Device:** In the prototype, this device acts as a simulated sensor node. It generates sensor data randomly, capturing the essence of real-world IoT devices. This data is then channeled through IoTA Streams, replicating the pathway a genuine sensor node would undertake.

2. **IOTA Streams:** A multifaceted module, the IOTA Streams is made up of several key entities that ensure its smooth operation. These include Data Publishers, who are responsible for disseminating data, Subscribers, who receive this data, Channels, which act as conduits for data flow, and additional features like Branching, Keyloads, and sequencing. Each of these entities, as explained in

Chapter 2.5.1, plays a critical role in ensuring secure and structured data transmission.

3. **Swarm Storage Network:** For the prototype's purpose, a bee node was employed as the storage medium within the Swarm network. This node liaises with the IOTA Streams, enabling both data transmission and retrieval. Advanced techniques, such as data chunking, are used for efficient data storage. Similarly, specialized retrieval mechanisms are employed to fetch the data when required, ensuring data integrity and speedy access.

4. **Consumer Node:** The endpoint of the architecture, this node, is equipped with a software stack that facilitates the retrieval of stored data. Once fetched, this data can be harnessed for diverse applications, depending on the end user's requirements. Whether it's for analytics, monitoring, or any other application, the Consumer Node is pivotal in making the stored data actionable.

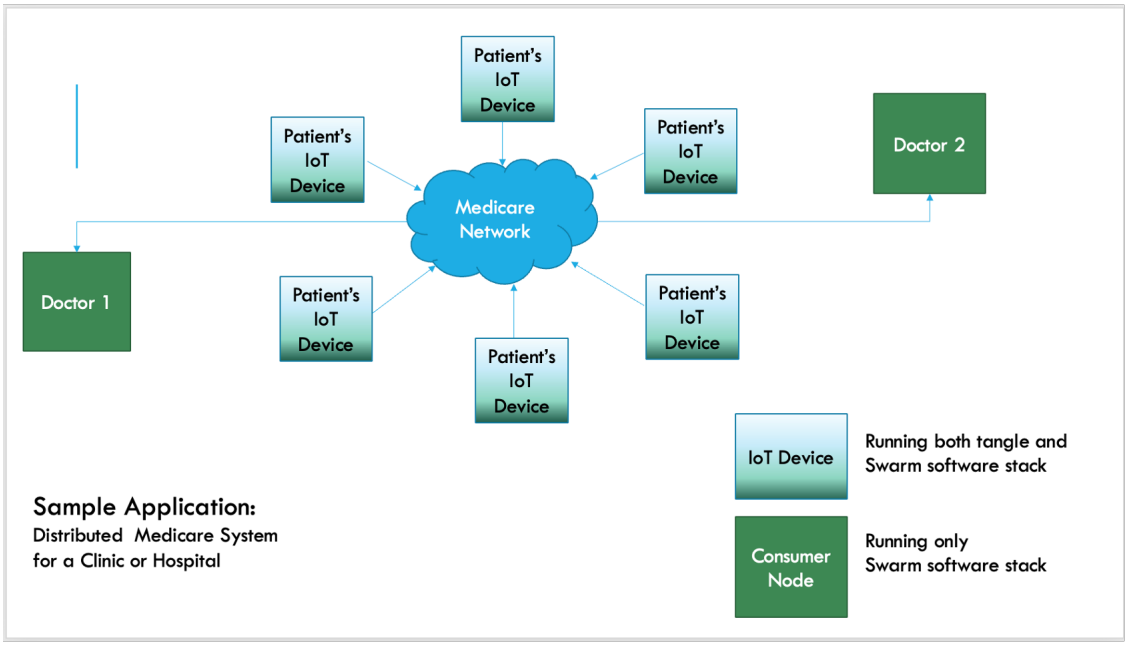


Figure 3.2: Sample Use case for the model proposed

Figure 3.2 presents a practical use-case scenario illustrating the potential applications of our proposed decentralized data model. Let's imagine a contemporary hospital setting that leverages the advantages of this model:

In this hospital, patients are equipped with IoT devices tailored to monitor various vital statistics such as temperature, blood pressure, heart rate, and more. These aren't just ordinary devices; they are connected entities that can relay real-time health metrics. Now, given the critical nature of these data points, their transmission must be both secure and efficient. Utilizing the proposed model, as soon as a patient's IoT device captures health metrics, it initiates the data transfer process. Doctors or medical staff, who are subscribed to receive this data, can access it almost instantaneously. This ensures that in critical situations, medical professionals can make informed decisions swiftly, potentially saving lives.

Our system uses IOTA Streams, which have cryptographic integrity checks in place. Any update in data would result in a new cryptographic signature, and interested parties could verify the data's integrity based on this new signature. Furthermore, the model isn't just about real-time data access. It also emphasizes data permanence and security. The data, once sent, gets stored in a tamperproof decentralized storage system. Given the sensitive nature of medical records and the increasing risks of data breaches in the digital age, such a storage system ensures that patients' health records are both accessible to authorized personnel and protected from any malicious threats.

In essence, this hospital scenario is a glimpse into the future of healthcare – where real-time data access complements robust data security, all powered by the decentralized model we propose.

3.2.2 IOTA Streams Integration

Integrating IOTA Streams into the data model enables secure data communication and encryption. Data is transferred and secured over an immutable distributed ledger called Tangle.

IOTA Tangle is not designed to be a long-term storage solution, but rather a secure and efficient transaction processing and data transfer network. So for long-term storage, an additional layer of data storage often needs to be considered. So, here we are using Swarm storage for the long-term persistence of data.

1. **Message Structure:** IOTA Streams utilizes a concept called "Channels" to organize and structure the data being streamed. Each channel contains a sequence

of "Messages." A Message consists of two main components: the "Header" and the "Payload."

- **Header:** Contains metadata and cryptographic information, including encryption keys and access control policies, required for securely handling the message.
- **Payload:** The actual data being transmitted within the message.

2. **Access Control Policies:** Access control policies in IOTA Streams are crucial for managing data privacy and determining who can access and read the messages within a given channel. The Access control policies are typically defined and enforced using cryptographic techniques.

   - **Signature-Based Access Control:** Each channel participant can be allocated unique cryptographic keys. Participants must sign messages in the channel using their private keys before reading or publishing them, and other participants can verify the signatures using the associated public keys.
   - **Asymmetric Encryption:** Asymmetric encryption can be used to encrypt the payload. Each participant has a distinct public-private key combination, and only those who have the correct private key may decrypt and read the payload.
   - **Role-Based Access Control:** Channels can also specify certain roles and the access rights that go with them. Participants can be allocated to distinct roles, giving them fine-grained access control.
   - **Revocation Mechanism:** If a participant's access has to be revoked, cryptographic key updates and re-encryption can be used to do it securely.

3. **Encryption Methods:** IOTA Streams employs various cryptographic techniques to ensure data privacy and integrity:

   - **Asymmetric Encryption**: For secure communication between participants and encrypting payload data.

- **Symmetric Encryption**: Used for efficient bulk data encryption within the channel.
- **Digital Signatures:** Used to verify the authenticity and integrity of messages and ensure they are from legitimate sources.
- **Hash Functions:** For ensuring data integrity and detecting any tampering with the messages.

Additionally, I've implemented sample code for the various types of communication possible in channels in different scenarios. Where users may choose between single and multiple publishers.

- **Single Publisher:**
  1. **Public Single Branch:** The most straightforward application of Streams. The author creates a public channel that anybody with the Announce message link can read.
  2. **Private Single Branch:** A private branch with restricted access. Subscribers must correctly subscribe to have access to the messages published by the Author.
  3. **Public Single Depth:** A retrievable public index channel. The author creates a public channel that anybody with the Announce message link can read. Messages can be retrieved by subscribers by utilizing an anchor message link and message number.
  4. **Private Single Depth:** A private index retrievable channel with user access restrictions. Subscribers must correctly subscribe to have access to the messages published by the Author. Subscribers can retrieve communications once they have been authorized by utilizing an anchor message link and message number.
- **Multi Publisher:**
  1. **Single Publisher per Branch:** The author creates a channel in which each subscriber is assigned their own branch to publish. This is accomplished by delivering a new Keyload to the channel for each new Subscriber, to which they may then connect their messages.

2. **Multiple Publisher per Branch:** The author creates a channel with two subscribers in each of the two branches. Subscribers A and B post their messages in branch A in alternating sequences, exhibiting the synchronization between each publishing entity to maintain state. The same is done in Branch B for Subscribers C and D.

- **Mixed Access Multi Branch:** A more advanced implementation of a Multi Branch channel with a mix of private and public access. Three branches are created, each with its own message chain and access limitations. Subscribers are defined as follows:

  1. **Subscriber A:** Has traditionally subscribed to and been allowed access to branch A (but can also read public branch C).
  2. **Subscriber B:** Can read from branch B using a Pre Shared Key (can also read from public branch C).
  3. **Subscriber C:** Is not properly subscribed and can only read from public branch C.

3.2.3 Swarm Storage Integration

Integrating Swarm Storage as the underlying decentralized storage layer for the data model provides distributed storage and increases the scalability of the data model which in turn helps in offering enhanced data availability, redundancy, and access control capabilities. I have configured and deployed a bee node in my implementation of the data model which in turn helps ensure data redundancy by replicating chunks across several nodes and the redundancy enables fault tolerance ensuring that data remains accessible even if some nodes fail or go down.

Swarm employs encryption methods to ensure the integrity and confidentiality of data.

3.2.4 Why IOTA Streams with Swarm?

IOTA Streams is a framework for publishing and consuming encrypted and authenticated data streams on the IOTA Tangle. It provides a way to store, retrieve,

and update data in a decentralized, immutable, and tamper-evident way. However, it doesn't specify a particular storage backend, which means that you can choose to store the data on the IOTA Tangle or some other storage platform.

Swarm, on the other hand, is a decentralized storage platform that is designed to provide distributed storage and retrieval of files and data objects. It is a peer-to-peer network that allows users to store and retrieve data in a decentralized way, without relying on any centralized servers or intermediaries. Thus, using IOTA Streams with Swarm, one can take advantage of the benefits of both platforms and create a more resilient and robust data storage and retrieval system.

3.3 Implementation

In this section, we discuss how we implemented the proposed architecture, taking into consideration the model's two essential components, IOTA Streams and Swarm Storage network. As shown in Figures 3.1 3.2, IoT devices run both the IoTA Tangle network and swarm network codes, and in turn, these IoT devices communicate data to IoTA Tangle using IoTA Streams which navigate and secure the data to Tangle, from which we send the data to the Decentralized storage network Swarm (Bee node).

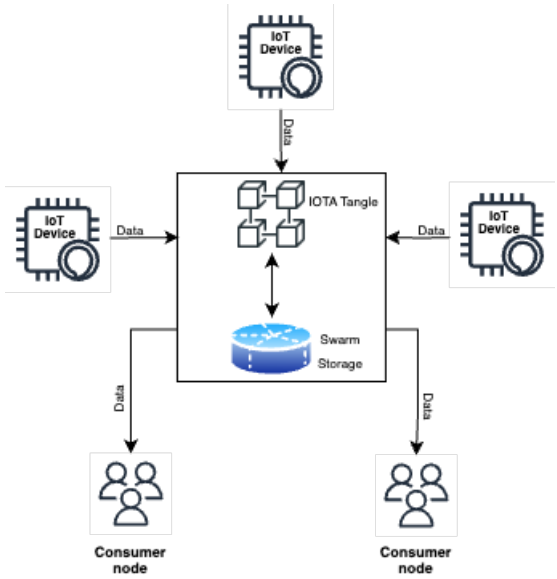


Figure 3.3: System Architectural Diagram of an end-to-end decentralized data model

3.3.1 IOTA Streams Implementation

- **Setup:** Install @iota/streams/node in the environment using npm to enable the iota streams package to work in the code. Also, install 'crypto' with npm to generate the seed for our nodes.

  COMMANDS: npm install @iota/streams/node , npm install @crypto

```
const streams = require("@iota/streams/node");
```

Figure 3.4: IoTA Streams package importing in code

Here, I used a public gateway node provided by the IOTA foundation to utilize the IOTA tangle framework. Ideally, we can create our own private gateway node to tangle to make it more secure and fast. So, I've used "https://chrysalis-nodes.iota.org/" as my node gateway.

```
// Node settings
let node = "https://chrysalis-nodes.iota.org/";
let options = new streams.SendOptions(node, 9, true, 1);
```

Figure 3.5: IoTA Node Settings

- **Data Streaming:** Various steps need to be implemented in the tangle for Data Streaming in IoTA Streams before the user may send actual data through.

  First, we need to establish an Author and a new Channel here I used a public single-branch channel model for which we will generate a new seed using the crypto package and sha256 hash generator for the author and a new channel address.

```
const crypto = require('crypto')

function createSeed() {
  const seed = crypto.createHash('sha256').update(crypto.randomBytes(256)).digest('hex');

  return seed;
}
```

Figure 3.6: Creating New Seed

```
Step 1
Author: Create author and new channel
Author seed:  d2dc33e1d0fbc15bc9321a06d6967720cf0c7989c3bb81f6fdc0585f3227ff46
Channel address:  f9531f32c67d89444f902c17dded80a0b42b1808835d394930f02cd417c1e4d2000000000000000
Multi branching:  false
```

Figure 3.7: Creating Author and Channel

The author then sends an announcement to the channel with an announcement link which is a combination of the channel address and additional IDs that subscribers will use to subscribe to the author's channel.

```
Step 2
Author: Send announcement
Announcement link:  f9531f32c67d89444f902c17dded80a0b42b1808835d394930f02cd417c1e4d20000000000000000:a5f62f44b2301bc900062600
https://explorer.iota.org/mainnet/message/5adb529d175ad917647fa074e5bea1f2b9fb89a34bb8022bec18957e608a53ca
```

Figure 3.8: Sending Announcement Link

Then we create a subscriber with a new seed using the same package that we used to generate the author seed 'crypto' and sha256. The subscriber will then get an announcement that we sent in the above step from the author and subscribe to the announcement link, for which the Subscription Link will be created.

```
Step 3
Subscriber: Create subscriber
Subscriber seed:  f1bb45d01055371f3665203f2ad92c468e4d3e79257d08bb5f54b61090786dec
```

Figure 3.9: Creating Subscriber

```
Step 4
Subscriber: Receive announcement and subscribe to channel
Subscription link:  f9531f32c67d89444f902c17dded80a0b42b1808835d394930f02cd417c1e4d20000000000000000:9c7602cf79baa9ac19540f17
https://explorer.iota.org/mainnet/message/5ae3777a19fb1e4b177b0af9417618d3a68ceebc5ef53245a9b7553e4807225a
```

Figure 3.10: Received announcement and Subscription Link

Now that the Author and Subscriber have been created and are ready for communication, the Author will receive a subscription link for which the Author will send a keyload message (A keyload message is a control and access restriction message that allows the author to indicate who should be allowed to decode any

messages attached after it. There are two methods for specifying access when constructing a keyload message: subscriber public keys and pre-shared keys.)

```
Step 5
Author: Receive subscription and send keyload message
Keyload link:  6cc9f26dfdb3bf65934e459aca716c818705026402b1f6720b9ee198b1df9093000000000000000000:890f50ff1f21df235adf3f28
https://explorer.iota.org/mainnet/message/c892539de12943ccb4dd1231ca5c44bb0af5a2152d4fd88c61819f2bf4974075
```

Figure 3.11: Received Subscription link and send Keyload message

The subscriber will now synchronize the channel state and send a tagged packet, resulting in the creation of a tagged packet link.

```
Step 6
Subscriber: Synchronize channel state and send tagged packet
Tagged packet link:  6cc9f26dfdb3bf65934e459aca716c818705026402b1f6720b9ee198b1df9093000000000000000000:31eeca627e855ed7e3b19aac
https://explorer.iota.org/mainnet/message/a93869db558d506cf55ff28b7897c4f01f1dece6e6fcde3e30b633434a1a9f03
```

Figure 3.12: Synchronizing Channel State and Sending Tagged packet

When the Author retrieves new messages in the channel, the author will receive a message link, which is a tagged packet link made up of tagged messages, which can be transmitted with a public or masked payload.

```
Step 7
Author: Fetch new messages from channel
Message link: 6cc9f26dfdb3bf65934e459aca716c818705026402b1f6720b9ee198b1df9093000000000000000000:31eeca627e855ed7e3b19aac
https://explorer.iota.org/mainnet/message/a93869db558d506cf55ff28b7897c4f01f1dece6e6fcde3e30b633434a1a9f03
Public payload:  dateTime: 25/07/2023 13:26:16, data: 89
Masked payload:  dateTime: 25/07/2023 13:26:16, data: 89
```

Figure 3.13: Fetching New Messages from Channel

Following that, I experimented with sending multiple signed packets from the subscriber, and the author will synchronize and fetch the messages.

```
Step 8
Subscriber: Synchronize channel state and send multiple signed packets
Signed packet #1 link:  6cc9f26dfdb3bf65934e459aca716c818705026402b1f6720b9ee198b1df9093000000000000000000:ea4defd216902c4f6638c6a6
https://explorer.iota.org/mainnet/message/3ee8edf0583b1fef4189dae4bd49943fd4112cf1dda04206bf854c5b0886805f


Signed packet #2 link:  6cc9f26dfdb3bf65934e459aca716c818705026402b1f6720b9ee198b1df9093000000000000000000:ea37d0ac7b13be21ad530171
https://explorer.iota.org/mainnet/message/1ce64bfc6517a752f0951b6ae22a6302d165e396d1db6d1e4c1653464475406b


Signed packet #3 link:  6cc9f26dfdb3bf65934e459aca716c818705026402b1f6720b9ee198b1df9093000000000000000000:f88a0e907ee4c922b705f618
https://explorer.iota.org/mainnet/message/e2badf39a0a7b5bea7a9086624d483b56899fd63012b613ef479d50b63144364
```

Figure 3.14: Sync channel state and send multiple signed packets

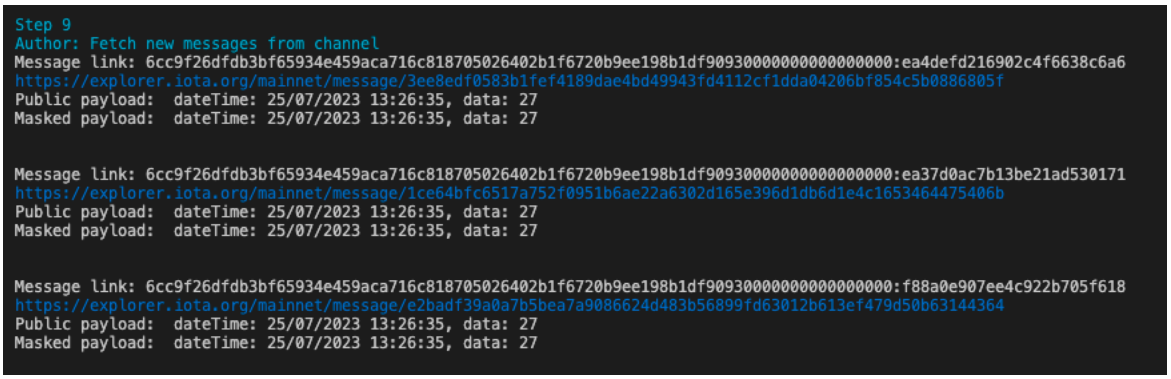


Figure 3.15: Fetching multiple new messages from Channel

3.3.2 Swarm Storage Implementation

In this part, we will discuss how we implemented the swarm storage network shown in Figures 3.1 3.3 IoT devices use a software architecture that combines IoTA Streams and Swarm Storage. IoTA Streams will interact with swarm storage.

- **Setup:** Using npm install commands, we first install the necessary swarm-related dependency packages in our environment, such as @ethersphere/bee-js, which will allow all of the methods and functions accessible in bee storage, also known as swarm storage.

```
const { Bee } = require('@ethersphere/bee-js');
```

Figure 3.16: Bee Storage package importing in code

Ideally, we should construct our own bee node in the system and use its address for communications and storage, but for prototype reasons, I utilized the publicly existing bee node "https://gateway.ethswarm.org".

```
// Initialize Bee instance
const bee = new Bee('https://gateway.ethswarm.org');
```

Figure 3.17: Bee node instance

- **Data Storage:** In both the author and subscriber code, we initialize the bee instance. We declare get and set methods in beeStorageAdapter that employ key and value pair encryption for uploading and downloading data.

```
// Initialize Bee instance
const bee = new Bee('https://gateway.ethswarm.org');
const beeStorageAdapter = {
  get: async (key) => {
    const data = await bee.downloadData(key);
    return data.text();
  },
  set: async (key, value) => {
    const { reference } = await bee.uploadData(key, value);
    //console.log("Swarm hash:", reference);
    return reference;
  },
};
```

Figure 3.18: Bee Storage Adapter Module

- **Integrating with Iota Streams** As we discussed in section 3.3.1 IoTA implementation is when the subscriber syncs and sends a tagged packet that is when we use the BeeStorage adapter function(Swarm) that we created to upload data and download data to fetch the messages.

```
//Store public payload on Swarm
let { reference } = await beeStorageAdapter.set(messageId, publicPayloadString);
console.log("Swarm hash:", reference);
```

Figure 3.19: Bee Integration with IoTA Streams

The above implementations are done with a single author and subscriber to implement the prototype with IoTA Streams and Swarm storage.

As stated in sections 3.2.2 and 3.3.1, IoTA Streams communicate between author and subscriber using a method known as channels. I explored many sorts of stream communications to look into new applications and other combinations of security mechanisms as stated below.

- **Single Publisher:**

1. **Public Single Branch:** The most straightforward application of Streams. The author creates a public channel that anybody with the Announce message link can read.

Figure 3.20: Public Single Branch Single Publisher Communication

2. **Private Single Branch:** A private branch with restricted access. Subscribers must correctly subscribe to have access to the messages published by the Author.

Figure 3.21: Private Single Branch Single Publisher Communication

3. **Public Single Depth:** A retrievable public index channel. The author creates a public channel that anybody with the Announce message link can read. Messages can be retrieved by subscribers by utilizing an anchor message link and message number.

Figure 3.22: Public Single Depth Single Publisher Communication

4. **Private Single Depth:** A private index retrievable channel with user access restrictions. Subscribers must correctly subscribe to have access to the messages published by the Author. Subscribers can retrieve communications once they have been authorized by utilizing an anchor message link and message number.

Figure 3.23: Private Single Branch Single Publisher Communication

- **Multi Publisher:**

1. **Single Publisher per Branch:** The author creates a channel in which each subscriber is assigned their own branch to publish. This is accomplished by delivering a new Keyload to the channel for each new Subscriber, to which they may then connect their messages.

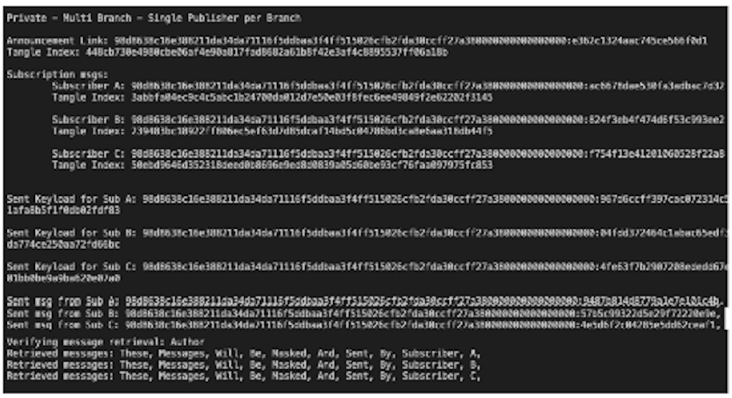


Figure 3.24: Private Multiple Branch Single Publisher per branch communication

2. **Multiple Publisher per Branch:** The author creates a channel with two subscribers in each of the two branches. Subscribers A and B post their messages in branch A in alternating sequences, exhibiting the synchronization between each publishing entity to maintain state. The same is done in Branch B for Subscribers C and D.

Figure 3.25: Private Multiple Branch Multiple Publisher per branch communication

- **Mixed Access Multi Branch:** A more advanced implementation of a Multi Branch channel with a mix of private and public access. Three branches are created, each with its own message chain and access limitations. Subscribers are defined as follows:

  1. **Subscriber A:** Has traditionally subscribed to and been allowed access to branch A (but can also read public branch C).
  2. **Subscriber B:** Can read from branch B using a Pre Shared Key (can also read from public branch C).
  3. **Subscriber C:** Is not properly subscribed and can only read from public branch C.

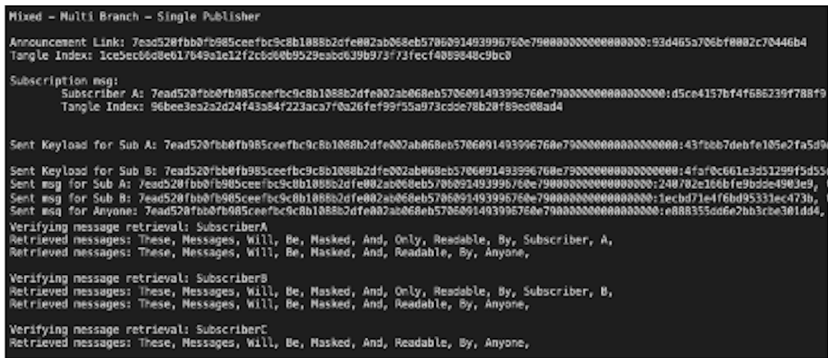


Figure 3.26: Mixed Multi Branch Single Publisher communication

Chapter 4

Experimental Results and Evaluation

4.1 Evaluation

The evaluation of the proposed end-to-end decentralized data model is a crucial step to assess its performance, scalability, security, and practical applicability in real-world scenarios. In this section, we present the evaluation methodology, experimental setup, and results obtained during the evaluation process.

The central aim of the experimental setup was to evaluate the efficiency and performance of the end-to-end decentralized data model leveraging Iota Streams and Swarm Storage. Several key metrics were measured, including data latency, system throughput, storage efficiency, and data integrity.

4.1.1 Data Latency

Understanding and minimizing data latency is of paramount importance in our research. This latency has a direct influence on the system's responsiveness, the reliability of data exchanges, and, ultimately, the user experience. In decentralized architectures such as the one we've developed using IOTA Streams and Swarm Storage, the efficacy of real-time data transfers becomes even more crucial. It's not just about speed; it's about maintaining the robustness and integrity of the entire system.

Unoptimized latency can lead to bottlenecks, potential data losses, and reduced trust in the system's efficiency. Therefore, our meticulous approach to measuring latency, down to the millisecond across a plethora of tests, is an essential step. This ensures that our decentralized data model isn't just theoretically sound but also practically efficient, ready to meet the stringent requirements of contemporary decentralized applications.

Complete Execution Time

In the entire operational sequence, the total execution time captures the period from the initial establishment of an author to the final phases of data packet transmission and subsequent retrieval. As elaborated in section 3.3.1, this encompasses several integral steps: the initialization of the author, setting up the channel and subscriber, the dispatch and reception of both the announcement and subscription links, ensuring synchronization, and the successful transmission and subsequent retrieval of data packets. Each of these stages is pivotal in comprehensively understanding the data flow within the system.

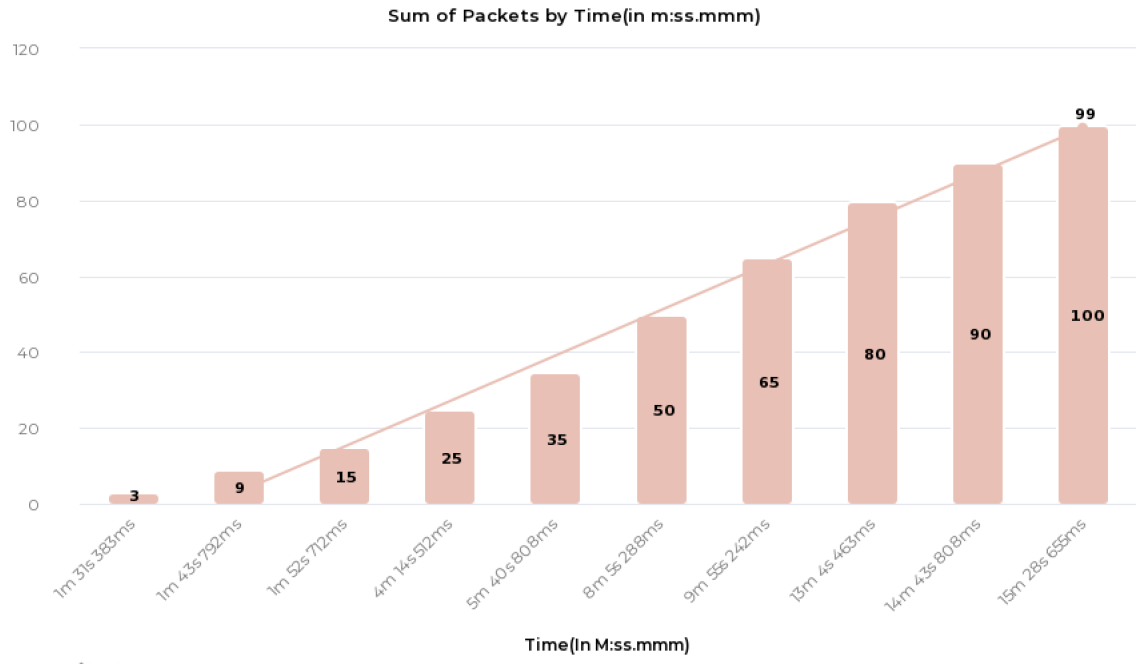


Figure 4.1: No of Packets vs Execution Time

From our illustration in 4.1, there's a clear correlation between the increase in packets and the total execution time of the program. As we dissect the graph, it becomes evident that with an uptick in the number of packets, there's a corresponding linear growth in the execution time. The trend line in the graph initiates at 9 packets and consistently grows linearly up to 99, within the range of 3 to 100 packets.

Turning our attention to the subsequent visualization below in 4.2, it offers a detailed perspective on how packets are dispersed with respect to their processing time. This particular bubble chart not only gives insights into individual packet

timings but also accentuates the overarching trend. It provides a normalized trend line that succinctly contrasts the number of packets with their associated processing time, offering a holistic view of the system's performance dynamics.

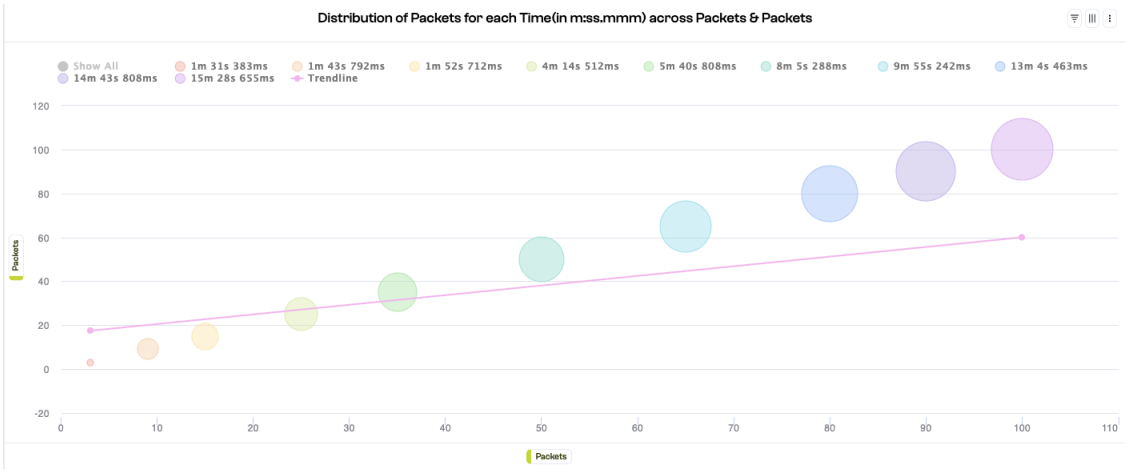


Figure 4.2: Distribution of Packets for each Time across Packets

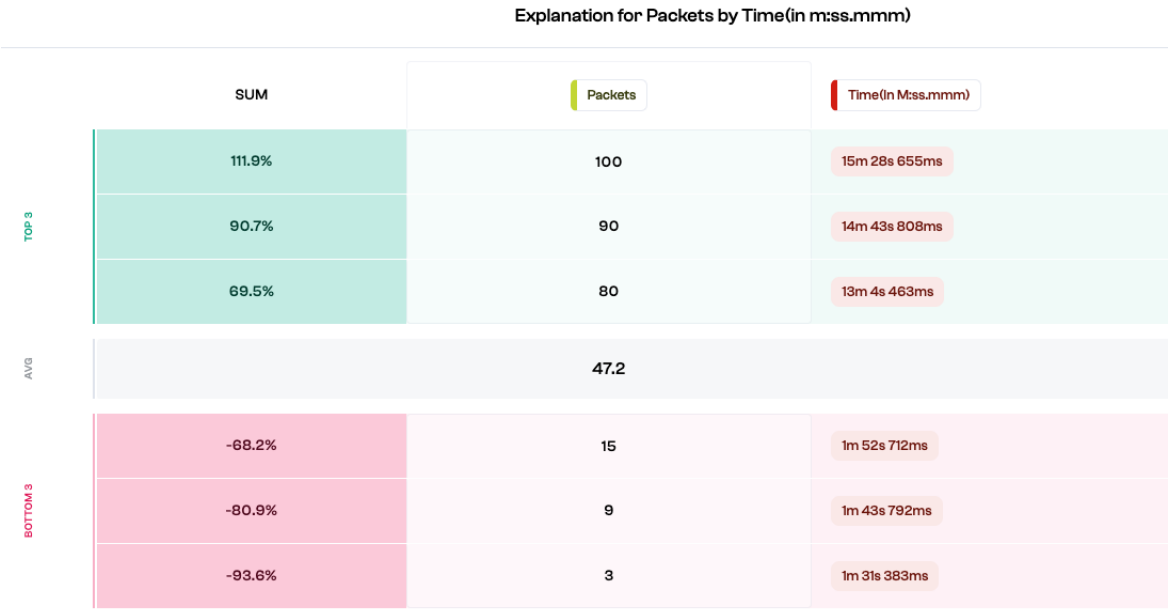


Figure 4.3: Outliers showing the Average number of Packets transmitted per time interval

The above figure 4.3 explains the outliers in respective of showing an average number of packets transmitted per time interval. As seen in the figure below the average number of packets is 47.2 for which I have illustrated the top and bottom 3

outliers showcasing how much difference in percentage of time with respective to an average number of packets.

In our examinations, it became evident that the total execution time is not consistent across varying scenarios—it is directly influenced by the intricacy of the tasks being performed and the volume of data subjected to processing. The latency manifested during these operations is more than just a time measure; it serves as a litmus test for the adeptness of our decentralized framework. By analyzing this latency, we can pinpoint potential areas of congestion or inefficiency that might hamper the system's optimal performance. This knowledge is instrumental in fine-tuning our model to ensure seamless data exchanges and bolster overall system responsiveness.

Data Exchange - Sending and Retrieving packets

The sending and retrieval of packets are core operations in our decentralized data model. We measured the latency of these operations for different numbers of packets ranging from 3 to 100. From our observations, a clear pattern emerged. As delineated in 4.4, there's a proportional relationship between the volume of packets and the time required for their transmission and retrieval. Simply put, the more packets involved, the longer the duration. Such a correlation is logical, considering that with an increase in packet numbers, the system is burdened with additional data that needs meticulous processing and transfer. This underscores the inherent challenges in managing voluminous data traffic in decentralized frameworks.

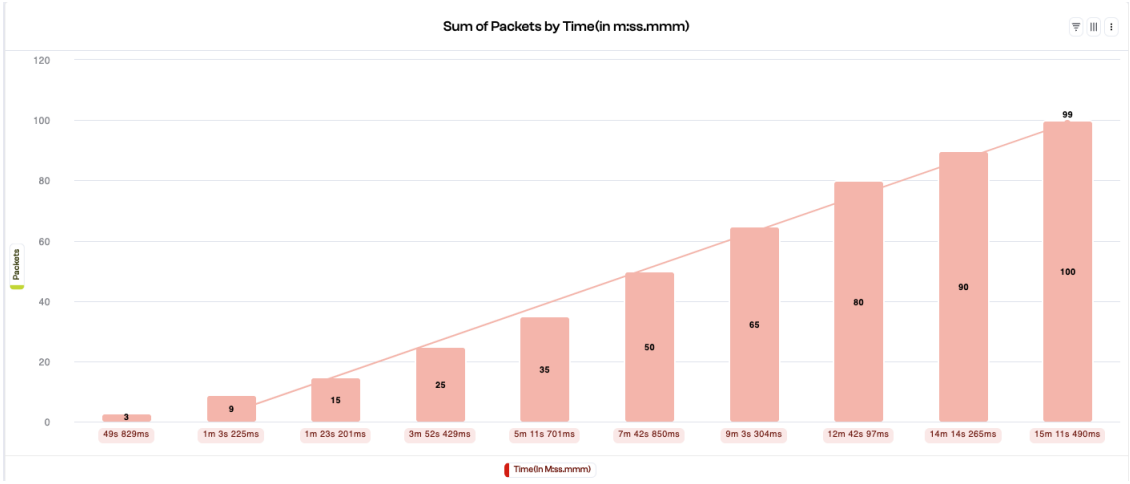


Figure 4.4: No.of Packets vs Time

In the subsequent bubble chart presented below 4.5, a discernible pattern emerges regarding the distribution of packets relative to the time taken for processing. The data portrays a linear progression, signifying that as the volume of data escalates, there is a proportional increase in the time required for its processing. Essentially, this trend underscores a fundamental principle: larger datasets invariably necessitate more processing time. This linearity accentuates the inherent relationship between data volume and processing latency, a vital consideration when evaluating the efficiency and scalability of decentralized systems.

Figure 4.5: Distribution of packets for each Time

In the subsequent section, we delve into anomalies or outliers concerning time about the average packet count, as depicted below 4.6. Through our observations, a particular discrepancy was noted. Specifically, when transmitting 80 packets, the time taken was an astounding 69.5 percent above the average time needed for sending 47.2 packets. Contrastingly, there was a significant reduction, amounting to 68.2 percent below the average, when the task involved transmitting just 15 packets.

These findings underscore a critical implication for applications that predominantly deal with low packet transmissions. Such applications stand to gain substantially from our model, benefiting from both speed and efficiency. Furthermore, this model serves as an enabler, facilitating communication in a wholly decentralized manner, especially in scenarios where lower packet counts are the norm.

Figure 4.6: Explanation of Packets by Time

We also observed a nonlinear relationship between the number of packets and the latency, which suggests that factors other than the sheer number of packets might affect the latency. These could include network congestion, hardware limitations, or the inherent computational complexity of the operations.

Below is a graph that 4.7 shows the difference in the time it took to execute all the steps against the time took for only packet transfer. Execution Time in the below graph includes the time it took to steps like creating of publisher, subscriber, channel, access control policies, and processing of packets. Where as in Packet execution time it only calculates the time it took for processing packets.

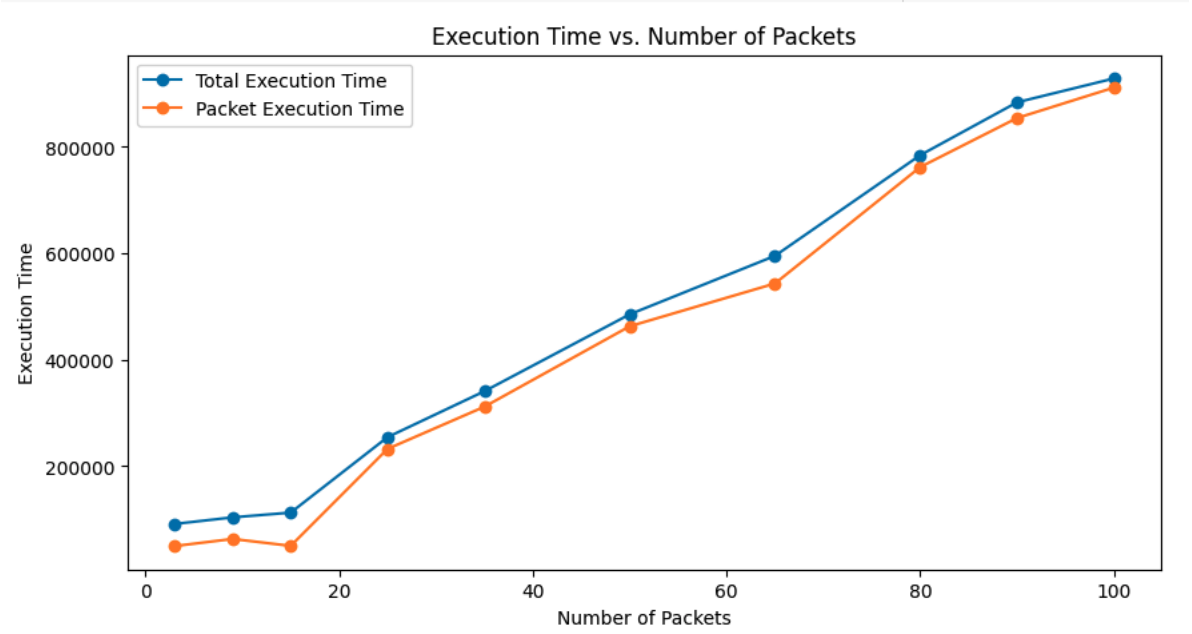


Figure 4.7: Time for Complete execution and Packet execution

On average, latency using the Iota Streams and Swarm storage architecture was the same as the typical centralized data models for low packet size transfers. This indicates the proposed decentralized system's capacity to deliver real-time or near real-time data exchange, an imperative in today's growing Internet of Things (IoT) infrastructure.

4.1.2 System Throughput

System throughput refers to the measure of how many units of information the system can process in a given amount of time. For our study, we observed the throughput by sending varying quantities of packets through the system and assessing the CPU usage and time taken to process these packets.

CPU Usage Analysis

To quantify the impact of our decentralized data model on the system resources, we recorded the CPU usage in percentage terms for each instance of the packet transmission.

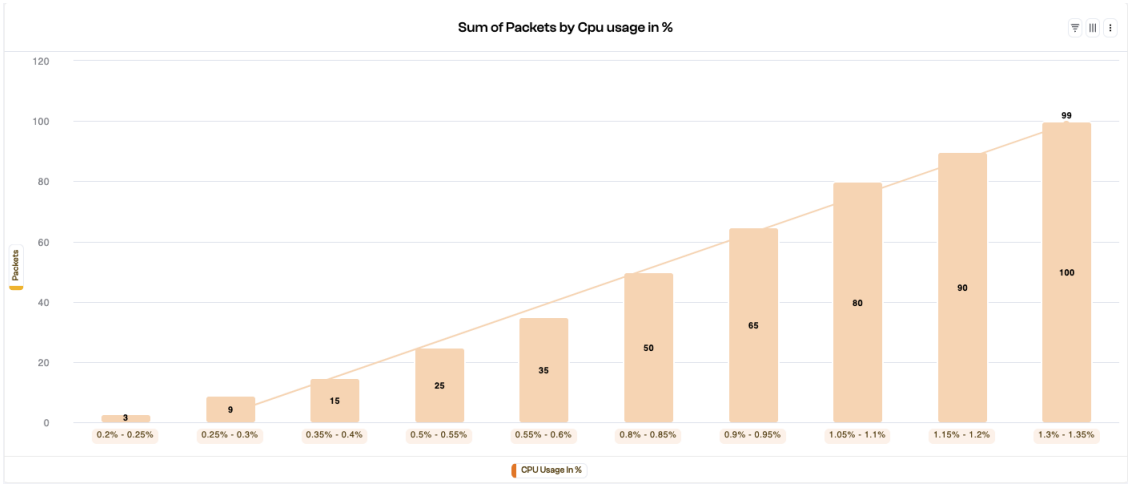


Figure 4.8: CPU Usage vs Packets

Our observations from the above 4.8 suggest that with an increase in the number of packets, CPU usage also increases. This is likely due to the additional computational resources required to process a higher number of packets.

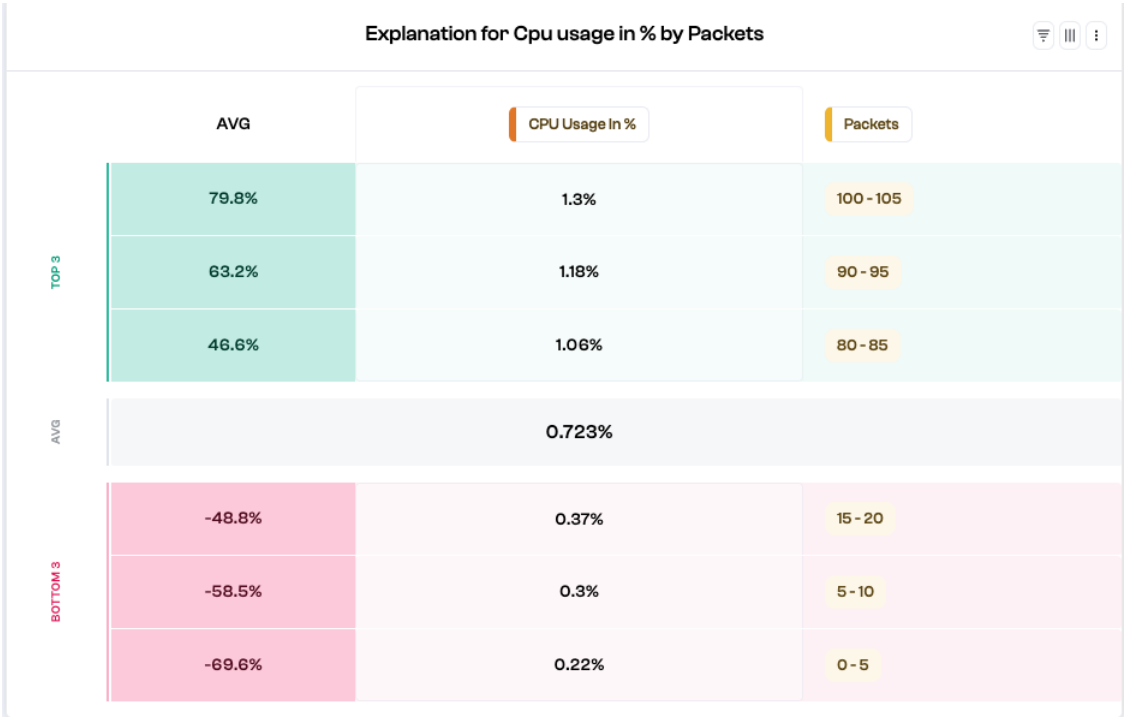


Figure 4.9: Explanation of CPU Usage by Packets

The insights provided by the above figure 4.9 allow for a deepened understanding of the system's behavior with respect to CPU usage as the number of packets handled

increases. The comparative analysis between the actual average CPU usage for a particular number of packets and the overall average CPU usage reveals intriguing findings.

In one of our observations, we noticed that for a range of 15-20 packets, the CPU usage averages around 0.37 percent. This is significantly lower - by 48.8 percent than the average CPU usage of 0.723 percent. This indicates that handling fewer packets (in this case, 15-20) considerably reduces the strain on CPU resources. Such efficiency in CPU utilization, while processing lower packet ranges, can contribute towards optimizing our system for scenarios where data loads are relatively smaller.

On the other hand, when the system deals with a heavier data load, specifically in the 80-85 packets range, it results in a CPU usage of approximately 1.06 percent. This is a substantial 46.6 percent higher than the overall average CPU usage of 0.723 percent. The increase in CPU usage under heavier load conditions demonstrates that our system's resource utilization scales with the number of packets processed, though at an escalated rate compared to the average. Understanding this behavior is critical when considering larger data transmissions or heavier workloads.

These observations become visually clear in the heatmap of CPU utilization against packets, as depicted in the subsequent figure 4.10. This heatmap portrays an evident progressive rise in CPU consumption as the size of the packet increases, underscoring the correlation between packet size and resource utilization. Interestingly, the CPU utilization seems to plateau in the 85-105 packet range, suggesting that the system may reach a point of maximum resource usage under our tested conditions. Further testing would be needed to confirm this.

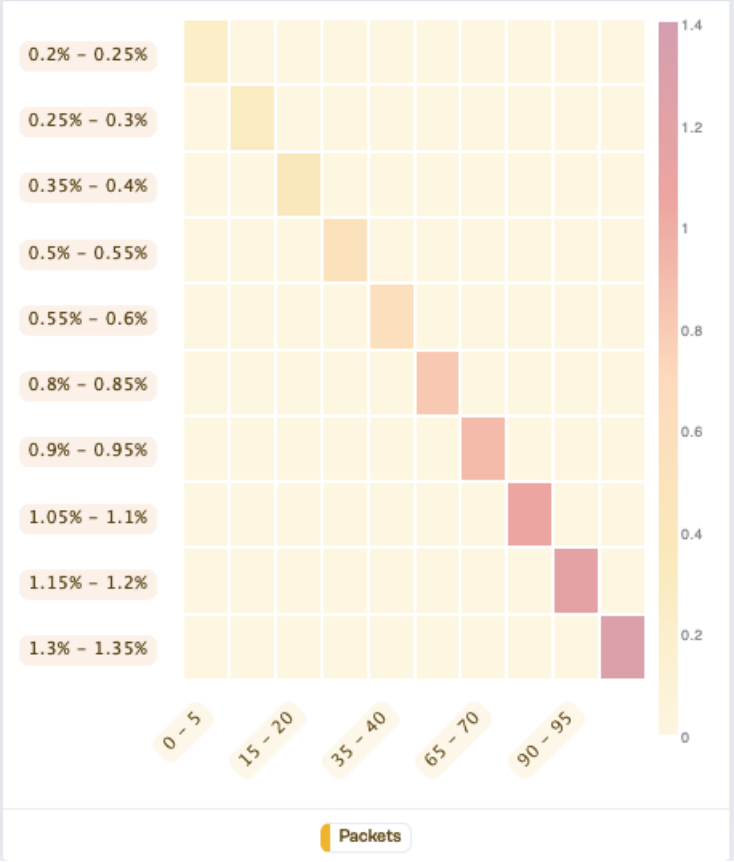


Figure 4.10: Heatmap of CPU Usage against packets

CPU User and System Usage Time

In addition to the overall CPU usage, we also measured the user and system CPU usage in milliseconds.

The user CPU time represents the amount of time the CPU was busy executing code in user mode, which in our case is the processing of packets. The system CPU time is the time the CPU was busy executing system-level operations, such as managing memory or I/O operations.

The graph below 4.11 provides a detailed illustration of our model's CPU utilization, revealing insights into user and system usage. Interestingly, we observed that the CPU system utilization is substantially lower compared to the user CPU usage. This suggests that our model predominantly runs user-level applications, minimizing its impact on system-level operations. Thus, it effectively leverages its resources, ensuring efficient performance. As workloads vary, continued observation of these

trends is crucial for optimizing future performance.

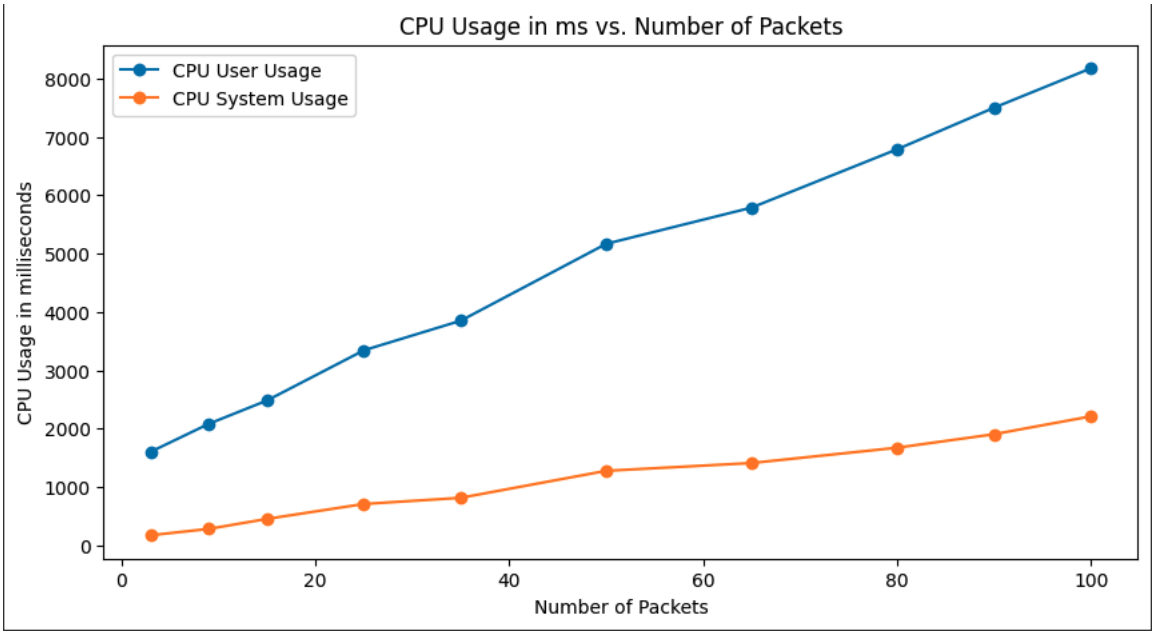


Figure 4.11: CPU Usage in ms against Number of Packets

Figure 4.12, a bar diagram below, illustrates a direct relationship between system CPU usage and the number of processed packets. As the packet count rises, system CPU consumption similarly increases. This amplification in system CPU usage is likely attributable to escalated system-level operations executed by the CPU, such as memory management and I/O operations. Each additional packet necessitates these operations, thereby driving up CPU consumption. This relationship not only underscores the system's scalability under increased loads but also helps in effective resource management and system optimization during high-data traffic scenarios.

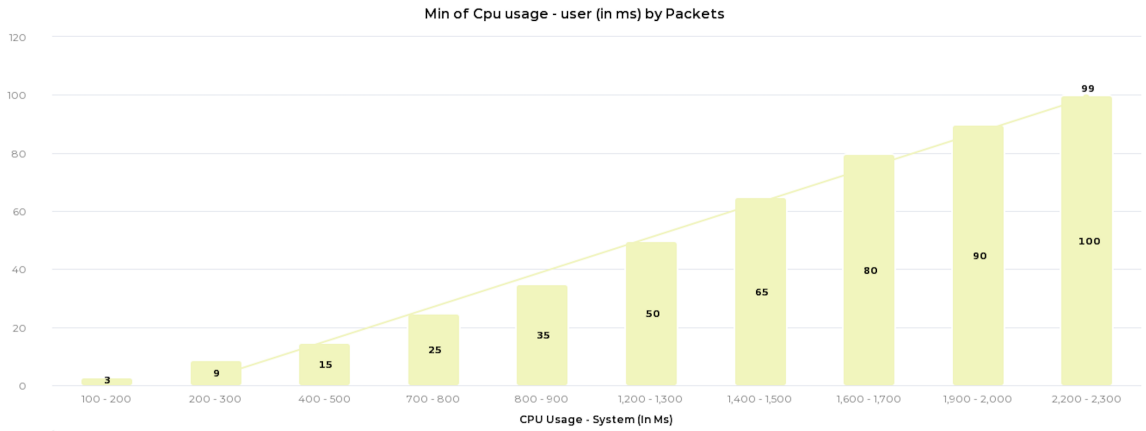


Figure 4.12: CPU User Usage in ms against Number of Packets

The forthcoming bar diagram 4.13 highlights the correlation between system CPU usage and the number of packets processed. It shows a clear upward trajectory, indicating that an increase in the number of packets directly influences the rise in system CPU usage. This surge is primarily attributable to the additional system-level operations required by the CPU. As the packet count increases, the CPU must execute more tasks such as memory management or I/O operations, leading to increased resource usage. Understanding this trend is crucial. It serves as an indicator of our system's performance under varying workloads, contributing towards strategic planning for future resource allocation and system optimization to ensure efficient operation even under high data traffic scenarios.

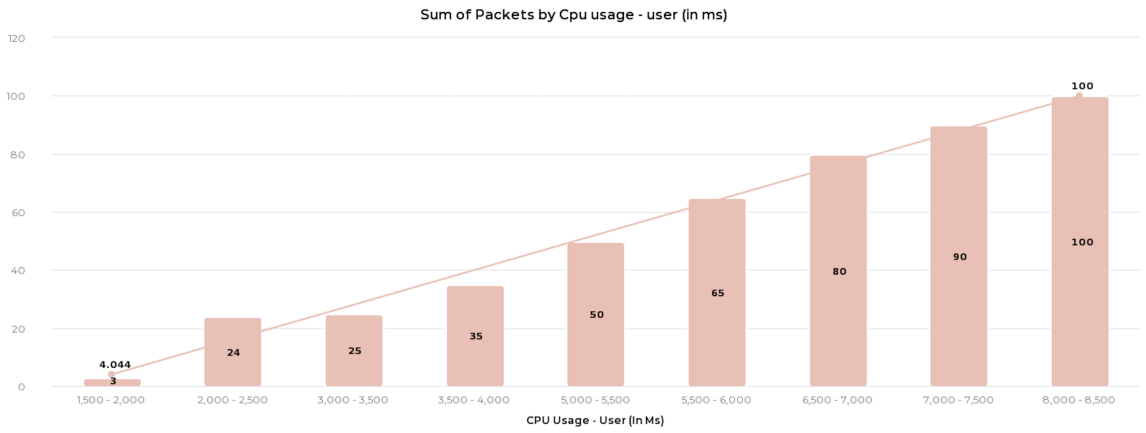


Figure 4.13: CPU System Usage in ms against Number of Packets

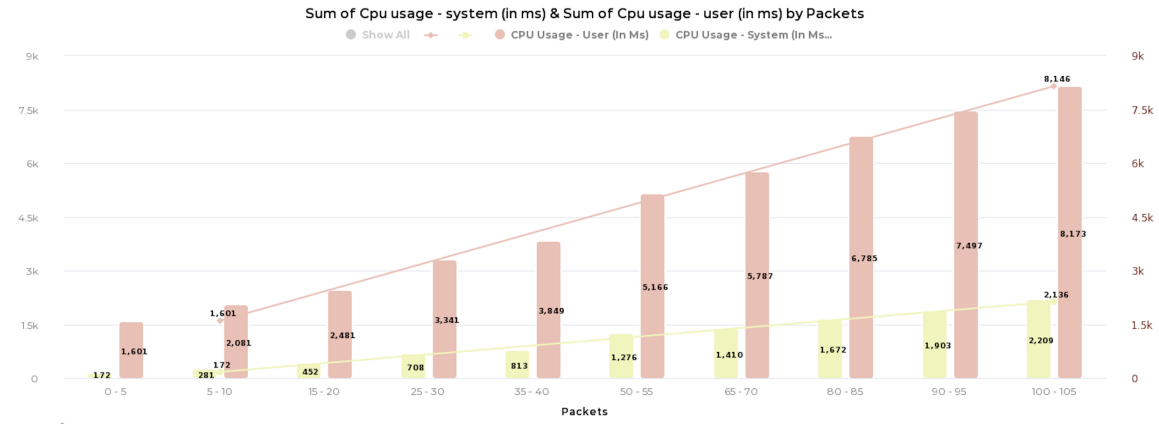


Figure 4.14: CPU System and User Usage in ms against Number of Packets

The data visualization provided above 4.14 represents a comprehensive view of the CPU usage for both user-level tasks and system-level operations in relation to the number of packets processed. The visualization includes trend lines that effectively demonstrate the correlation between CPU utilization and packet count.

An observation from these trend lines is the ascending nature of CPU usage as the number of packets escalates. However, it is noteworthy that the trend lines do not encompass the range of 0-5 packets. This absence suggests that when the data load is relatively low - that is when the data is transmitted in smaller packet sizes - the system exhibits significantly low throughput.

Understanding these trends and behavior helps anticipate the system's performance under varying workloads. It offers insights that can guide system optimizations for different data transmission scenarios, ensuring that the system can maintain an optimal balance between throughput and CPU utilization.

The table provided below 4.15 offers a comprehensive analysis of our data projections, utilizing the mean number of packets as a tool for an enhanced understanding of CPU usage patterns. By identifying the top and bottom three outliers in the data for CPU Usage in percent by packets, CPU User Usage, and CPU System Usage, we gain a clearer perspective on the system's behavior in extreme situations. The top three outliers may suggest conditions of unusual stress or high load, causing a spike in CPU usage. Conversely, the bottom three outliers could illustrate scenarios where the system is under minimal load, resulting in low CPU usage.

By computing these outliers, we understand the system's behavior under extreme

conditions better. This, in turn, helps us refine our understanding of the model's performance characteristics. For example, the top three outliers indicate scenarios where the system may have been under unusual stress or high loads, causing a significant spike in CPU usage. On the other hand, the bottom three outliers might represent scenarios where the system handled smaller loads with minimal CPU usage. The information derived from the outliers' analysis provides valuable insights into the robustness and resilience of the system.

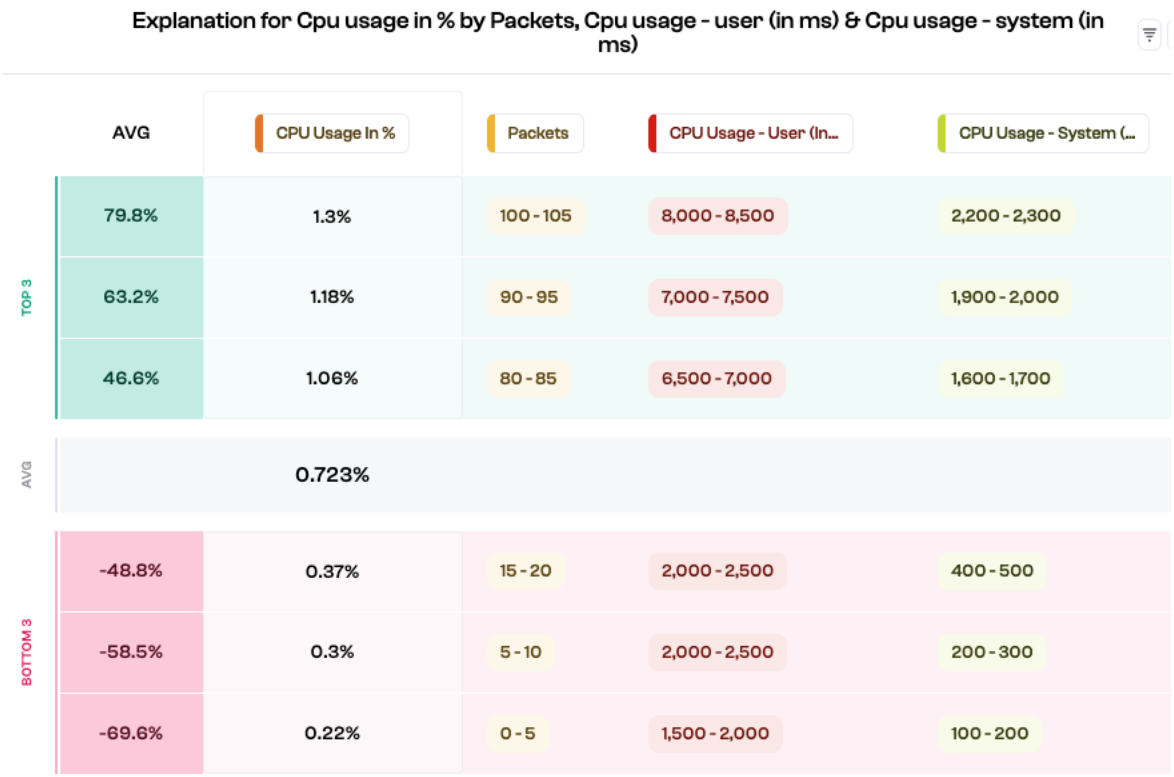


Figure 4.15: Average packets against CPU Usage in percent, packets, CPU user, and system usage

The data model showcased low latency and minimal resource consumption, indicating its suitability for real-time data processing, but this measure is for very simple application of sending simulated sensor data, performance can vary with different types of applications and length of the messages.

We investigated the data model's scalability by increasing the number of data publishers and subscribers in the network. As illustrated in section 3.3.2 we have implemented various types of communications possible with the model. The increase

in network participants had a minimal impact on data retrieval times and transaction throughput.

4.1.3 Data Integrity

Data integrity not only guarantees the trustworthiness of data over its entire life cycle but also protects the data from external alterations or modifications. When operating in a decentralized environment, where data is spread across numerous nodes, the risks of inconsistencies and vulnerabilities become more pronounced. As there isn't a single centralized authority to oversee and validate every piece of data, the importance of preserving integrity becomes paramount. Within our research framework, our assessment of data integrity wasn't just confined to Error detection and understanding the aftermath, i.e., detecting errors once they have occurred. We also delved deep into Error prevention(preemptive measures), aiming to prevent errors before they could even manifest. This dual-pronged approach is crucial in ensuring that data remains trustworthy and unaltered, no matter where it resides within the decentralized network

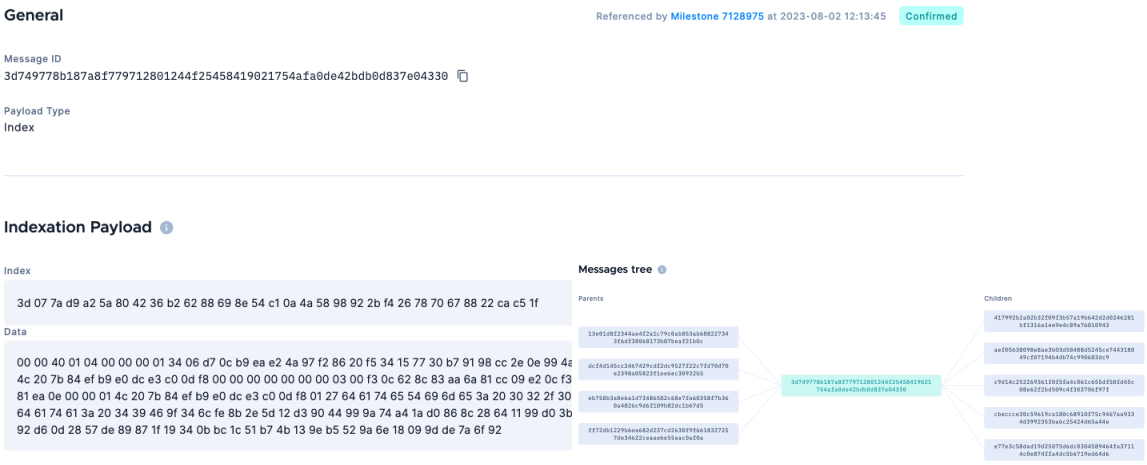
Error Detection

Error detection refers to the process of identifying any discrepancies that may have arisen post-data transmission. To test this aspect, we conducted numerous experiments wherein we transmitted data packets through our system and then compared the transmitted and received data.

For a comprehensive validation, we curated a wide range of test cases encompassing a diverse array of data types and structures. A crucial component of our error detection strategy was the utilization of checksum or hash comparisons. This involved creating a condensed representation or 'digest' of the transmitted data and comparing it with a similar 'digest' of the received data. This procedure is highly effective in verifying data integrity, as it can reveal even subtle alterations in the data that might otherwise go unnoticed.

```
Author: Fetch new messages from channel
Message link: 06d70cb9eae24a97f28620f534157730b79198cc2e0e994a007763d50d188dd70000000000000000:9589db186afe5fa13194011e
https://explorer.iota.org/mainnet/message/3d749778b187a8f779712801244f25458419021754afa0de42bdb0d837e04330
Public payload:  dateTime: 02/08/2023 15:13:36, data: 49
Masked payload:  dateTime: 02/08/2023 15:13:36, data: 49
```

Figure 4.16: A Hash of the payload on the tangle



(a) A Confirmation of message on the tangle **(b)** A Confirmation of Payload on the tangle

Figure 4.17: Data confirmation on IOTA Tangle

So, once the tangle transaction is approved, we can view the indexation payload on the tangle gateway as shown above in Figure 4.17, which has a data hash that we converted to text and cross-checked to see if there are any differences in the data that we truly transmitted.

Figure 4.18: Hash to text Conversion of payload

After comparing figures 4.16 and 4.18 we can infer that the data transferred and

hashed on the tangle are the same with extra padding for security purposes, and that there is no loss or alteration in data, indicating data integrity.

Error Prevention

Error prevention focuses on strategies that help avoid any inconsistencies or errors in the data in the first place. In our system, we employed mechanisms like data validation before transmission and reliable error handling during data processing.

Error Simulation

Here, we intentionally introduced errors in the data transmission and assess if the system can detect and handle them appropriately.

The screenshot below 4.19 depicts our validation code for the data and DateTime fields.

```
const validateJSON = function(json) {
  // Define your validation logic here

  // Check if 'data' is a number
  if (typeof json.data !== 'number') {
    console.error("Invalid data format. Data should be a number.");
    return false;
  }

  // Check if 'dateTime' is a string and matches the expected format
  if (typeof json.dateTime !== 'string' || !/^(\d{2}\/){2}\d{4} \d{2}:\d{2}:\d{2}$/.test(json.dateTime)) {
    console.error("Invalid dateTime format. dateTime should be a string in 'DD/MM/YYYY HH:MM:SS' format.");
    return false;
  }
```

Figure 4.19: Validation code snippet

We sent a dateTime value in a different format than what is expected and see if the system identifies the error as shown in figure 4.20.

```
Subscriber: Synchronize channel state and send multiple signed packets
/Users/krish/Desktop/Research-Sem/iota-streams/subscriber.js:144
      let json = {"data": randomNumber, "dateTime":dateTime};
                                                   ^
```

Figure 4.20: Modify the data payload

Added an extra field to the payload that our system isn't expecting and check if the program detects it and throws an error.

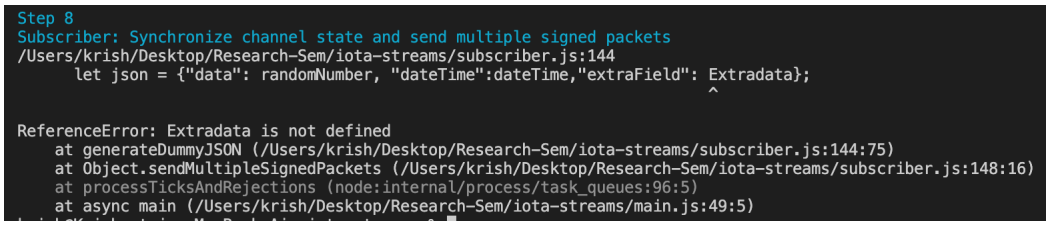


Figure 4.21: Extra Field detection

We incorporated IOTA streams and Swarm storage in our model being decentralized technologies they offer built-in security features that are incorporated into our model, IOTA Streams security aspects to end-to-end encryption, forward secrecy, Data confidentiality, and Data Integrity, Swarm offers storage level security like data sharding, chunk encryption, data authentication, decentralization, incentive mechanisms.

Chapter 5

Conclusion and Future Work

5.0.1 Limitations and Future Work:

While the evaluation yielded promising results, some limitations were observed. Further research can explore optimizations for even higher performance, investigate quantum-resistant cryptography for long-term security, and address interoperability challenges with other decentralized technologies.

Overall, the evaluation process confirms the success of our proposed end-to-end decentralized data model, offering a valuable contribution to the field of decentralized data management. The data model's robustness, efficiency, and versatility make it a promising solution for a wide range of industries, revolutionizing how data is managed, shared, and secured in decentralized networks.

5.0.2 Conclusion

In this thesis, we proposed and implemented an end-to-end decentralized data model leveraging the power of IOTA Streams and Swarm storage. The main objective of the research was to design and evaluate a secure, efficient, and scalable data management solution in a decentralized network. Throughout the study, we explored the principles, features, and challenges of decentralized data management, blockchain technologies, IOTA Streams, and Swarm storage.

The research findings demonstrate that the integration of IOTA Streams and Swarm storage in the proposed data model presents a promising approach to decentralized data management. The combination of IOTA Streams secure communication layer and Swarm storage's distributed storage capabilities empowers users with enhanced data privacy, tamper resistance, and availability. The architectural design of the data model allows data publishers to securely share information with authorized data subscribers, without the need for a central authority or intermediary.

Our implementation and performance evaluation revealed that the end-to-end decentralized data model showcases efficient data management capabilities with minimal resource overhead. The model demonstrated competitive transaction throughput and data retrieval time, making it a viable option for real-world decentralized applications that require fast and secure data handling.

The combination of IOTA Streams data encryption and access control mechanisms with Swarm storage's decentralized data replication and retrieval methods offers a robust and scalable solution for decentralized data management. However, we acknowledge some limitations in our research. As with any emerging technology, the implementation of IOTA Streams and Swarm storage may require additional testing and optimization in complex and large-scale scenarios. Furthermore, the security of the data model heavily relies on the strength of cryptographic algorithms, and advancements in quantum-resistant cryptography should be explored for long-term data protection.

In conclusion, this thesis contributes to the growing body of knowledge in decentralized data management and blockchain-based solutions. The end-to-end decentralized data model using IOTA Streams and Swarm storage provides an innovative and practical approach to secure and scalable data management in a decentralized network. It offers significant potential for use in various domains, including supply chain, IoT, finance, and identity management.

The research opens up avenues for future work, such as exploring integration with other decentralized technologies, conducting more extensive performance evaluations, and addressing challenges related to interoperability and regulatory compliance. As the decentralized landscape evolves, the proposed data model can serve as a foundation for further advancements in secure and decentralized data management solutions.

Overall, the results obtained from this research demonstrate the feasibility and effectiveness of the proposed end-to-end decentralized data model, positioning it as a valuable contribution to the field of decentralized data management and its potential applications in a wide range of industries.

Bibliography

[1] U. Sarfraz, S. Zeadally, and M. Alam, "Outsourcing iota proof-of-work to volunteer public devices," *Security and Privacy*, vol. 3, Dec 2019.

[2] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, p. 2787–2805, 2010.

[3] C. Liu, C. Yang, X. Zhang, and J. Chen, "External integrity verification for outsourced big data in cloud and iot: A big picture," *Future Generation Computer Systems*, vol. 49, p. 58–67, 2015.

[4] G. Zyskind, O. Nathan, and A. entland, "Decentralizing privacy: Using blockchain to protect personal data," *2015 IEEE Security and Privacy Workshops*, 2015.

[5] S. Popov, "The tangle," *White paper*, vol. 1, no. 3, p. 30, 2018.

[6] I. FOUNDATION, "Iota streams: Framework for building privacy-respecting and secure applications.," 2020.

[7] T. V, F. D., N. D.A, F. Z., and J. L, "Swap, swear, and swindle: Incentive system for swarm," *Swap, Swear, and Swindle: Incentive System for Swarm. Swarm Orange Papaer.*

[8] R. Roman, J. Zhou, and J. Lopez, "On the features and challenges of security and privacy in distributed internet of things," *Computer Networks*, vol. 57, no. 10, p. 2266–2279, 2013.

[9] E. Schiller, A. Aidoo, J. Fuhrer, J. Stahl, M. Ziörjen, and B. Stiller, "Landscape of iot security," *Computer Science Review*, vol. 44, p. 100467, 2022.

[10] N. Živi, E. Kadušić, and K. Kadušić, "Directed acyclic graph as tangle: an iot alternative to blockchains," pp. 1–3, 2019.

[11] J. K. Parmar and K. G. Vaghani, "A conceptual study on holochain and blockchain technology," *Artificial Intelligence and Communication Technologies*, p. 331–341, 2022.

[12] B. Cao, Z. Zhang, D. Feng, S. Zhang, L. Zhang, M. Peng, and Y. Li, "Performance analysis and comparison of pow, pos, and dag-based blockchains," *Digital Communications and Networks*, vol. 6, no. 4, pp. 480–485, 2020.

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

[14] Y. Luo, Y. Chen, Q. Chen, and Q. Liang, "A new election algorithm for dpos consensus mechanism in blockchain," in *2018 7th International Conference on Digital Home (ICDH)*, pp. 116–120, 2018.

[15] S. M. Skh Saad and R. Z. Raja Mohd Radzi, "Comparative review of the blockchain consensus algorithm between proof of stake (pos) and delegated proof of stake (dpos)," *International Journal of Innovative Computing*, vol. 10, Nov. 2020.

[16] F. Wang, Y. Ji, M. Liu, Y. Li, X. Li, X. Zhang, and X. Shi, "An optimization strategy for pbft consensus mechanism based on consortium blockchain," in *Proceedings of the 3rd ACM International Symposium on Blockchain and Secure Critical Infrastructure*, BSCI '21, (New York, NY, USA), p. 71–76, Association for Computing Machinery, 2021.

[17] J. Song, F. Bai, Y. Zhu, T. Shen, and A. Xie, "An improved-poa consensus algorithm for blockchain-empowered data sharing system," in *Proceedings of the 2022 4th Blockchain and Internet of Things Conference*, BIOTC '22, (New York, NY, USA), p. 128–134, Association for Computing Machinery, 2022.

[18] M. Scherer, "Performance and scalability of blockchain networks and smart contracts," 2017.

[19] D. Khan, L. T. Jung, and M. A. Hashmani, "Systematic literature review of challenges in blockchain scalability," *Applied Sciences*, vol. 11, no. 20, 2021.

[20] K. Sharma and D. Jain, "Consensus algorithms in blockchain technology: A survey," in *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pp. 1–7, 2019.

[21] M. Schäffer, M. di Angelo, and G. Salzer, "Performance and scalability of private ethereum blockchains," in *Business Process Management: Blockchain and Central and Eastern Europe Forum* (C. Di Ciccio, R. Gabryelczyk, L. García-Bañuelos, T. Hernaus, R. Hull, M. Indihar Štemberger, A. Kő, and M. Staples, eds.), (Cham), pp. 103–118, Springer International Publishing, 2019.

[22] A. A. Monrat, O. Schelén, and K. Andersson, "Performance evaluation of permissioned blockchain platforms," in *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*, pp. 1–8, 2020.

[23] E. Karaarslan and E. Konacakli, *Data Storage in the Decentralized World: Blockchain and Derivatives*, pp. 37–69. 12 2020.

[24] N. Zahed Benisi, M. Aminian, and B. Javadi, "Blockchain-based decentralized storage networks: A survey," *Journal of Network and Computer Applications*, vol. 162, p. 102656, 2020.

[25] S. Wilkinson, J. Lowry, and T. Boshevski, "Metadisk a blockchain-based decentralized file storage application," *Storj Labs Inc., Technical Report, hal*, pp. 1–11, 2014.

[26] D. Trautwein, A. Raman, G. Tyson, I. Castro, W. Scott, M. Schubotz, B. Gipp, and Y. Psaras, "Design and evaluation of ipfs: A storage layer for the decentralized web," SIGCOMM '22, (New York, NY, USA), p. 739–752, Association for Computing Machinery, 2022.

[27] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.

[28] D. Vorick and L. Champine, "Sia: Simple decentralized storage," *Retrieved May*, vol. 8, p. 2018, 2014.

[29] D. P. Bauer, *Filecoin*, pp. 97–101. Berkeley, CA: Apress, 2022.

[30] C. C. Agbo, Q. H. Mahmoud, and J. M. Eklund, "Blockchain technology in healthcare: A systematic review," *Healthcare*, vol. 7, no. 2, 2019.

[31] G. Van Norman, "Decentralized clinical trials: The future of medical product development?," *JACC. Basic to translational science*, vol. 6, pp. 384–387, 04 2021.

[32] shindemangesh, "Blockchain in network marketing. how is blockchain used in marketing? - mangesh shinde website," Aug 2020.

[33] A. N. Kudtharkar, N. S. Bidarkundi, P. Geethika, L. Kamoji, D. G. Narayan, and P. Shettar, "Attribute based access control for cloud resources using smart contracts," in *2023 International Conference on Networking and Communications (ICNWC)*, pp. 1–5, 2023.

[34] iotaledger, "Secure digital infrastructure - get started — iota https://www.iota.org/solutions/secure-digital-infrastructure."

[35] W. F. Silvano, D. De Michele, D. Trauth, and R. Marcelino, "Iot sensors integrated with the distributed protocol iota/tangle: Bosch xdk110 use case," in *2020 X Brazilian Symposium on Computing Systems Engineering (SBESC)*, pp. 1–8, 2020.

[36] H. Shah, "Building industrial iot with iota: Introduction and how iota works," Jan 2019.

[37] A. Rawat, V. Daza, and M. Signorini, "Offline scaling of iot devices in iota blockchain," *Sensors*, vol. 22, p. 1411, Feb 2022.

[38] iotaledger, "streams/documentation/docs/overview.md at main ·iotaledger/streams."

[39] ethersphere, "Github - ethersphere/swarm: Swarm: Censorship resistant storage and communication infrastructure for a truly sovereign digital society," Jan 2021.

[40] S. Sheridan, K. Sunderland, and J. Courtney, "Swarm electrification: A comprehensive literature review," *Renewable and Sustainable Energy Reviews*, vol. 175, p. 113157, 2023.

[41] J. Hartman, I. Murdock, and T. Spalink, "The swarm scalable storage system," pp. 74 – 81, 02 1999.

[42] E. Daniel and F. Tschorsch, "Ipfs and friends: A qualitative comparison of next generation peer-to-peer data networks," *IEEE Communications Surveys Tutorials*, vol. 24, no. 1, pp. 31–52, 2022.

[43] M. I. Khalid, I. Ehsan, A. K. Al-Ani, J. Iqbal, S. Hussain, S. S. Ullah, and Nayab, "A comprehensive survey on blockchain-based decentralized storage networks," *IEEE Access*, vol. 11, pp. 10995–11015, 2023.

[44] M. Swan, *Blockchain: Blueprint for a new economy.* " O'Reilly Media, Inc.", 2015.

[45] M. Naz, F. A. Al-Zahrani, R. Khalid, N. Javaid, A. M. Qamar, M. K. Afzal, and M. Shafiq, "A secure data sharing platform using blockchain and interplanetary file system," *Sustainability*, vol. 11, p. 7054, Dec 2019.

[46] M. I. Khalid, I. Ehsan, A. Al-Ani, J. Iqbal, S. S. Ullah, and Nayab, "A comprehensive survey on blockchain-based decentralized storage networks," *IEEE Access*, vol. PP, 01 2023.

[47] H. R. Hasan, K. Salah, I. Yaqoob, R. Jayaraman, S. Pesic, and M. Omar, "Trustworthy iot data streaming using blockchain and ipfs," *IEEE Access*, vol. 10, p. 17707–17721, 2022.

[48] O. Lamtzidis and J. Gialelis, "An iota-based distributed sensor node system," in *2018 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–6, 2018.