UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Computer Science

Product Tracing in the Norwegian Fishing Industry Supply Chain Utilizing GoQuorum Blockchain and Smart Contracts

Erik Godtliebsen

INF-3981 Master's Thesis in Computer Science – June 2023

List of Tables

List of Figures

Acknowledgements

I would like to express my sincerest thanks to my supervisor, Professor Håvard Dagenborg. His guidance, insightful feedback, and encouragement throughout this thesis have been essential. To my friends whom I have studied and played table tennis with these past five years, I express my heartfelt thanks. Your friendship and competitive spirit made this journey memorable. Lastly, my deepest gratitude goes to my parents for the constant encouragement and loving support I have received.

Abstract

The Norwegian fishing industry faces a significant issue of fishery crimes, with product traceability systems presenting a potential solution to counter these illegal activities. Current supply chain management in the seafood industry is vulnerable to information alterations, thereby complicating product traceability. Blockchain technology, with its unique properties, offers an interesting approach to address these challenges. Despite this, existing blockchain-based product traceability systems often fail to integrate government regulation and provide limited access to traceability data for consumers. Moreover, those providing such access often lack user-friendliness. This thesis explores if a blockchain-based product traceability system can support supply chain management, enhance consumer confidence, and enforce regulatory compliance. We conducted a review of existing literature and assessed the potential of blockchain technology to optimize supply chain management. Furthermore, a traceability system, entitled SeaChain, incorporating a permissioned blockchain, smart contracts, and governmental regulations was developed. We evaluated this system and compared it with existing systems. Our findings suggest that blockchain technology can enhance supply chain management, bolster consumer trust, and aid in mitigating fishery crimes. The research conducted provides valuable insights for improving supply chain management and contributes to future studies in this field.

1 Introduction

Norway holds a prominent position as one of the world's leading seafood-exporting nations. In 2022, the country exported nearly 3 million tons of seafood, generating revenues of approximately 150 billion NOK [1]. The fishing industry plays a crucial role in the Norwegian economy, providing a significant source of income and employment opportunities. Moreover, the exported seafood serves as an essential food supply for countries that import Norwegian seafood products. Unfortunately, the fishing industry faces challenges posed by illegal fishing activities. Fishery crimes have significant consequences, leading to substantial economic losses, over-exploitation of marine resources, and environmental degradation. Additionally, these crimes threaten global food security by obscuring the origin and processing methods of products [2].

One potential solution to mitigate fishery crimes is the implementation of a product tracking system incorporating provenance data throughout the fishing industry supply chain. Such a system can not only help ensure regulatory compliance by businesses but also enhance consumer confidence in food products. Supply chains can be quite complex, often involving a multitude of enterprises that collaborate and interact at various stages of the production and distribution process. In traditional supply chain management, data is primarily recorded by each enterprise in a centralized database to which only the respective enterprise has access [3]. This not only allows for easy data manipulation, which can be exploited to serve the enterprise's interests, but it also poses a significant obstacle for government entities attempting to monitor and detect illegal activities within the supply chain. Consequently, it leads to mistrust between enterprises and results in inconsistent information throughout the supply chain. Traditional product tracking systems face challenges due to these shortcomings in supply chain management.

Blockchain technology presents innovative approaches for tracking products within supply chains. Blockchain's decentralized, transparent, and immutable nature addresses the limitations of traditional supply chain management [3]. By integrating smart contracts into the blockchain network, various processes can be automated, eliminating the need for intermediaries. This combination of blockchain technology and smart contracts enable secure and immutable product tracing throughout the supply chain, ensuring no authority or enterprise can manipulate data.

In this thesis, a proof-of-concept, smart contract-based product traceability system, dubbed SeaChain, has been developed and tailored for the Norwegian fishing industry. SeaChain, built upon the GoQuorum blockchain, is designed to trace fish and associated products in batches using the smart contracts. The GoQuorum network comprises supply chain organizations and the Directorate of Fisheries, a government-controlled regulatory body. SeaChain enhances consumer confidence by providing QR codes on products, which can be scanned to view comprehensive transaction histories and provenance data.

1.1 Problem definition

Fishery crimes in the Norwegian sea pose significant challenges to the Norwegian fishing industry. A system capable of preventing or reducing these illegal activities would be highly beneficial. Blockchain technology and smart contracts have the potential to address these issues by enhancing the transparency, security, and data integrity of the supply chain.

Our thesis is:

"A blockchain-based traceability system utilizing smart contracts can address the limitations of traditional supply chain management by providing secure and transparent tracking, leading to enhanced consumer confidence and regulatory compliance."

To strengthen the thesis, our research will pursue the following objectives:

1. Implement a proof-of-concept product traceability system for the Norwegian fishing industry using blockchain technology and smart contracts.
2. Assess whether such a system can improve the Norwegian fishing industry supply chain and help reduce fishery crimes.

1.2 Context

This thesis is set within the context of the Cyber Security Group (CSG) at UiT The Arctic University of Norway. CSG is a research group that tackles fundamental challenges within the domain of distributed systems. Its primary objective is to develop robust methodologies for the design and implementation of reliable and efficient distributed systems. Furthermore, CSG focuses on the application of digital technology in various sectors, such as clinical medicine and financial fraud prevention [4, 5].

In recent years, CSG has been actively involved in numerous projects aimed at mitigating fraud in the fishing industry [6-8]. This thesis specifically explores the application of blockchain technology, a type of distributed system, with the goal of mitigating fishery crimes. The focus of this work naturally aligns with the ongoing research interests and objectives of CSG.

1.3 Method

In 1989 the Task Force on the Core of Computer Science released a report on how to divide the discipline of computing into three major paradigms. These paradigms are theory, abstraction, and design [9]. An explanation of each paradigm is provided below.

The theory paradigm, rooted in the principles of mathematics, adheres to the following four-step process for developing a coherent and valid theory:

1. Definition – Characterize the objects of study.
2. Theorem – Formulate hypotheses about potential relationships among the identified objects.
3. Proof – Evaluate the theorems, proving or disproving them.
4. Interpretation – Interpret the results obtained from the proof stage.

The abstraction (or modeling) paradigm, which is based on the experimental scientific method, follows this four-step process to investigate a phenomenon:

1. Hypothesis – Formulate a hypothesis about the phenomenon being researched.
2. Model & Predict – Construct a model based on the hypothesis and make a prediction about its behavior or outcomes.
3. Experiment & Data Collection – Design an experiment to test the model and collect relevant data.
4. Analysis – Analyze the results from the experiment and data collection stage.

The design paradigm is based on engineering principles and uses the following four steps in the construction of a system intended to solve a given problem:

1. Requirements – State the requirements that the system needs to meet.
2. Specifications – Outline the detailed specifications that guide the system's design and functionality.

3. Design & Implementation – Design the system according to the stated specifications and implement it.
4. Testing - Conduct thorough testing to ensure the system functions as intended and meets the requirements and specifications.

In our research, we incorporate elements of the abstraction and design paradigms to investigate the problem posed in our thesis statement. We have formulated a hypothesis and made a prediction of its outcome. While we do not perform an experiment in a real-world scenario, we have designed and tested a product tracing system, which aims to address the challenges related to supply chain management and fishery crimes. Despite the lack of real-world experimentation, our methodology remains comprehensive, ensuring the system operates as intended.

1.4 Scope and limitations

We make two key assumptions regarding the product tracing system:

1. Each fishing vessel is equipped with an onboard system that automatically gathers data related to the species and weight of fish caught, as well as the GPS location of the boat.
2. All connections between applications are assumed to be secure, employing modern cryptographic protocols and standards, such as TLS, to ensure data integrity and confidentiality.

1.5 Outline

Section 2 reviews existing product tracing systems and explores the role of blockchain technology in supply chain management.

Section 3 provides the necessary theoretical background to understand the work presented in this thesis.

Section 4 presents the design and implementation of SeaChain, the product tracing system developed in this thesis.

Section 5 evaluates the performance of the components used in building SeaChain.

Section 6 conducts a security analysis of SeaChain, compares it to existing product tracing systems, and reflects on the achievements of this thesis.

Section 7 concludes the thesis and outlines potential areas for future work.

2 Related work

This section explores research related to the application of blockchain technology within supply chain management. The primary objective is to examine existing product tracing systems, as well as to study blockchain technology's role in supply chain management. The goal is to gain a comprehensive understanding of the research field and to establish a context for the work conducted in this thesis with respect to the existing literature. The primary criteria for selecting the reviewed papers in Table 1 include their recency and relevance to the use of blockchain in supply chain management. This section is structured to provide an overview of each selected paper individually. A comparative analysis between the reviewed product tracing systems and SeaChain will be presented in Section 6.2.

Authors	**Paper**	**Technology**
Wang *et al.* [3]	Smart Contract-Based Product Traceability System in the Supply Chain Scenario	Public blockchain, smart contracts
Ding *et al.* [10]	Permissioned Blockchain-Based Double-Layer Framework for Product Traceability System	Permissioned and private blockchain, smart contracts
Madumidha *et al*. [11]	A Theoretical Implementation: Agriculture Food Supply Chain Management using Blockchain Technology	Public blockchain, smart contracts, IoT
Malik *et al.* [12]	ProductChain: Scalable Blockchain Framework to Support Provenance in Supply Chains	Permissioned blockchain
Lin *et al.* [13]	Food Safety Traceability System Based on Blockchain and EPCIS	Public blockchain, smart contracts
Salah *et al*. [14]	Blockchain-Based Soybean Traceability in Agricultural Supply Chain	Public blockchain, smart contracts

Table 1: Overview of the reviewed product tracing systems

Wang *et al.* [3] designed a product traceability system built on top of the Ethereum blockchain for supply chain management. The system utilizes smart contracts to enable smooth information flow and trace transactions throughout the supply chain. To view the transaction history of a product, a consumer must join the blockchain network as a node and interact with a webpage that requires manual input. The authors proposed an event response mechanism to guarantee

the validity of a transaction by verifying the identities of both parties, but this mechanism was not implemented. Updating the transaction history of a product requires human interaction through a front-end webpage that interacts with the smart contracts. The system tracks materials and products separately but fails to create a link between them. Consequently, while a product can be traced back to its manufacturer, the origin of the materials used in the product remains untraceable.

Ding *et al.* [10] identify several issues with existing product traceability systems, such as a lack of consideration for government regulation, difficulties in protecting sensitive enterprise data, and performance bottlenecks. To address these challenges, they propose a product traceability scheme based on a permissioned blockchain within a double-layer framework. The primary layer comprises a consortium blockchain, while the secondary layer consists of private blockchains belonging to individual enterprises. These layers employ smart contracts to facilitate government regulation and update product traceability information. The two layers are connected by key nodes, which are responsible for maintaining both the consortium blockchain in the primary layer and the private blockchain in the secondary layer. The authors claim that their double-layer framework can reduce regulatory costs, protect sensitive data, improve performance of product data querying, enhance scalability, and ensure tamper resistance. However, the framework does not present a mechanism for consumers to access the product traceability data, resulting in no consumer accessibility.

Madumidha *et al*. [11] present a theoretical description of a system that leverages the Ethereum blockchain, smart contracts, and Internet of Things (IoT) technology to enable food traceability in the agriculture supply chain. The authors review major drawbacks of traditional supply chain management, including the lack of traceability and concerns surrounding food safety. They argue that implementing a food traceability system can enhance transparency, streamline management, and bolster trust between consumers and suppliers. As part of this system, they consider the use of an app to display product traceability data to consumers, thus increasing accessibility. Furthermore, they claim that traceability can help minimize errors and mitigate unethical and illegal activities.

Malik *et al.* [12] proposed a consortium-based blockchain framework, dubbed ProductChain, designed to enable consumers and stakeholders to trace the origin of products. The blockchain network consists of government regulatory bodies and the key entities within the supply chain. Featuring a three-tiered architecture, the system employs sharding to address scalability

concerns. The framework incorporates a transaction vocabulary that allows a final product to be linked to multiple raw ingredients, as well as access control mechanisms that ensure no single participant dominates the blockchain. The system leverages QR codes on products for consumers to retrieve traceability data. However, the interface used to display data is not showcased, leaving the accessibility to consumers somewhat unclear. Additionally, the authors conducted a security analysis demonstrating the system's resilience against a wide variety of client and network-based attacks.

Lin *et al.* [13] designed a food safety traceability system, leveraging blockchain technology and the Electronic Product Code Information Services (EPCIS). Their system enables consumers to query product data using a product code and a smart contract address. Unfortunately, the authors do not explain this process in detail or display what the consumer interface looks like. To mitigate the issue of data explosion arising from rapid accumulation of data in the blockchain, their system integrates dynamic management of on-chain and off-chain data. Smart contracts are employed to protect sensitive information and prevent data tampering. The authors developed a prototype utilizing the Ethereum blockchain and compared its performance with existing traceability systems. Their evaluation results demonstrate that the proposed system outperforms three others in terms of tamper-resistance, privacy protection, centralization, and the volume of data stored on-chain.

Salah *et al*. [14] present a versatile framework utilizing the Ethereum blockchain and smart contracts for tracking, tracing, and executing business transactions within the soybean agricultural supply chain. The framework is designed to remove the need for intermediaries and trusted centralized authorities. Its generic design enables the implementation of trusted and decentralized traceability for a wide range of crops and products in the agricultural supply chain. The authors claim that their system offers enhanced transparency and traceability in a secure, reliable, and efficient manner. Despite these advances, the framework does not provide a method for consumers to view traceability data for products. The framework also does not address critical challenges in blockchain technology, including scalability, governance, privacy, standards, and regulations. In future research, the authors aim to address these issues and incorporate proof of delivery and automated payments into their system.

In conclusion, this section has reviewed various studies that focus on blockchain-based supply chain management, particularly with respect to product tracing systems. The reviewed literature highlights key issues in supply chain management and demonstrates how the combination of

blockchain technology and smart contracts can improve transparency, traceability, and security. Furthermore, the studies suggest that food traceability systems can enhance management efficiency, foster trust between suppliers and consumers, and reduce illegal activities within the supply chain. Despite these benefits, it is evident that many existing product tracing systems still have issues related to scalability, governance, privacy, and regulations. By building upon the insights gained from the reviewed literature, this thesis aims to contribute to the advancement of product tracing systems by addressing some of these challenges. A comparative analysis between the reviewed systems and SeaChain will be presented in Section 6.2.

3 Background

This section provides the necessary theoretical foundation required for understanding the work presented in this thesis. It is designed to address any questions that may have arisen from Section 2 and to deepen the reader's knowledge of blockchain technology. After reading both Section 2 and this section, the reader should have a solid understanding of blockchain technology and its application in supply chain management. The section begins with an introduction to blockchain technology, followed by an explanation of the Ethereum blockchain, smart contracts, and decentralized applications. The focus then shifts to the GoQuorum blockchain, which is built on top of Ethereum and forms the backbone of the work presented in this thesis. Essential properties of consensus protocols are examined before delving into the specific consensus protocol used in this thesis. The section concludes with an explanation of product tracing, provenance data, and international standards for unique identifiers.

3.1 Blockchain

Blockchain technology is a type of Distributed Ledger Technology (DLT) characterized by decentralized databases operating as Peer-to-Peer (P2P) networks without a central authority. Users within the network share, replicate, and synchronize data using a consensus algorithm [15]. The term blockchain first emerged in the whitepaper for Bitcoin, the world's first cryptocurrency [16]. A cryptocurrency is a type of digital currency that uses cryptography for security and enables parties to transfer funds without relying on a financial institution [16, 17].

In a blockchain, transactions are grouped together and stored in blocks, which are then linked to one another using cryptographic hashes [15]. This structure creates an immutable chain of records that is infeasible to alter without controlling the majority of the nodes in the network. The first block created is the foundation of the blockchain and is denoted as the genesis block. Every block contains metadata, but this data may vary a lot depending on the blockchain platform. Typical metadata are block numbers, a hash to the previous block, and a timestamp for when the block was created. Figure 1 shows an example of how the blocks in Bitcoin are linked together. A block contains metadata and a list of its recorded transactions. The hash pointing to the previous block is created by using the SHA256 algorithm on the metadata of the previous block [16].

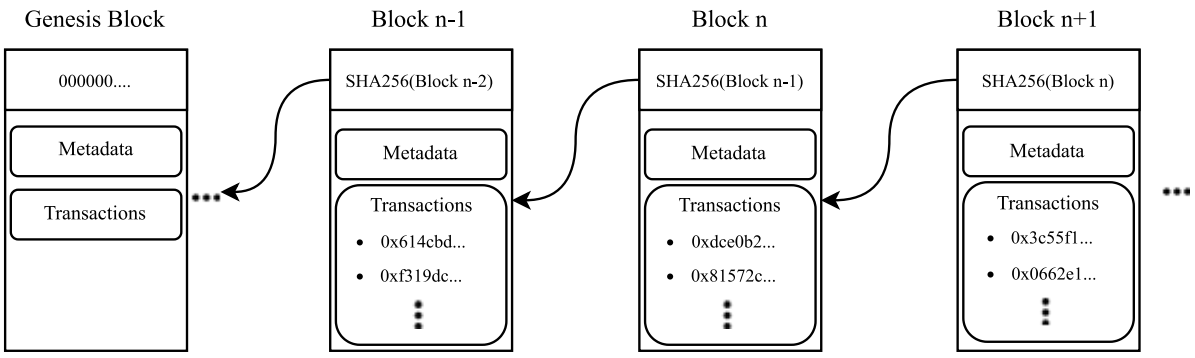

Figure 1: Bitcoin blockchain

The three main types of blockchains are public, private, and consortium [15, 18]. Public blockchains do not have admission control that regulates who can join the network. This means anyone can become a node in the network and read and write transactions. Private blockchains enforce strict access control and only allow authorized users to participate. Typically, a single authority controls the network and access to data is restricted to members. Private blockchains are more centralized than the public variants but still provide useful features such as data immutability and auditability. Consortium blockchains, also known as permissioned blockchains, are a hybrid between public and private blockchains, controlled by a group of organizations or entities. The governing group decides on the rules, permissions, and who can participate in the network. Consortium blockchains can be used when multiple parties need to collaborate without there being any third-party that is trusted by each member [18].

The choice of consensus algorithm for a blockchain is crucial, as it heavily impacts the performance of the blockchain. This includes metrics like transaction verification speed, transaction throughput, block creation speed, and scalability. Furthermore, the consensus algorithm affects security, energy efficiency, and accessibility [19]. Bitcoin and Ethereum are the most well-known public blockchains, and both initially used the consensus algorithm Proof of Work (PoW) [16, 20, 21]. However, as of 2022, Ethereum has switched to Proof of Stake (PoS), an algorithm that offers better energy efficiency, reduced hardware requirements, and increased security [22]. GoQuorum is an example of a blockchain that can be configured as either private or consortium-based. This blockchain supports multiple Proof of Authority (PoA) consensus algorithms, such as Clique and Istanbul Byzantine Fault Tolerance (IBFT) [23].

In summary, blockchains offer key properties such as decentralization, immutability, security, transparency, and consensus. These properties make blockchains suitable for use in various industrial sectors, including agriculture, healthcare, and supply chain management [15].

3.2 Ethereum

Ethereum is a decentralized blockchain platform featuring an embedded computer called the Ethereum Virtual Machine (EVM). Every node participating in the network maintains a copy of the EVM state, ensuring consensus on its current state. The EVM is Turing-complete, meaning it can execute any algorithm provided sufficient time and resources, enabling complex computations [20]. In comparison to Bitcoin, which primarily serves as a cryptocurrency platform, Ethereum offers a more modern and versatile blockchain solution.

To interact with the Ethereum network and send transactions, you must have an account. There are two types of accounts in Ethereum: Externally Owned Accounts (EOA) and Contract Accounts. Both types of accounts have a unique public address that serves as an identifier for the account. The account address is a 42-character hexadecimal string that starts with a "0x" prefix. An EOA can be controlled by anyone who possesses the private keys associated with the account. In contrast, a Contract Account is associated with a smart contract and is controlled by the code within the contract [22].

An Ethereum transaction is initiated by an EOA and contains instructions to be executed on the blockchain. Transactions can be categorized as regular transactions, contract deployment transactions, or contract execution transactions. A regular transaction could e.g. involve sending cryptocurrency from one account to another. Contract deployment transactions occur when a smart contract is deployed, while contract execution transactions refer to any transaction that interacts with a deployed smart contract [22]. Figure 2 shows an example of a transaction and transaction receipt from interacting with a smart contract. Table 2 provides an explanation of the various fields displayed in Figure 2.

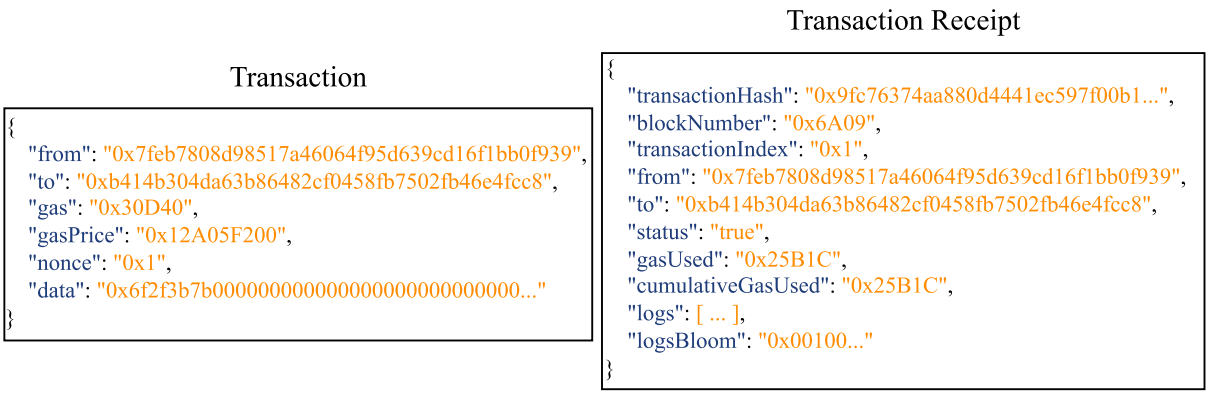

Figure 2: Transaction and transaction receipt in JSON format.

Field	Explanation
from	The Ethereum address of the account that sent the transaction.
to	The Ethereum address of the smart contract.
gas	The amount of gas the sender is willing to spend on the transaction.
gasPrice	The price of the gas for the transaction.
nonce	A number representing the total amount of transactions sent from the sender's address.
data	The input data for the transaction, e.g., a smart contract function call and the provided arguments.
transactionHash	A unique hash identifier for the transaction.
blockNumber	The block the transaction was included in.
transactionIndex	The index position of the transaction inside of the block.
status	A Boolean indicating if the transaction was successful or not.
gasUsed	The gas consumed by the transaction.
cumulativeGasUsed	The total amount of gas used in the block after the transaction was executed.
logs	An array containing log objects generated by the transaction. These log objects represent events that were emitted by the smart contract while executing the transaction.
logsBloom	A 256-byte bloom filter that is a compact representation of the logs included in the transaction receipt. The bloom filter enables efficient searching and filtering without processing the entire logs array.

Table 2: Explanation for the fields in Figure 2

In Ethereum, gas measures the amount of computational effort needed to execute specific operations on the blockchain. All transactions require computational resources, and therefore each transaction has a gas fee. These fees are paid in Ethereum's native currency, ether (ETH) [22]. The nodes that participate in the network and add transactions to the blockchain are compensated for their work with transaction fees, which are calculated based on the gas used and paid in ETH. The concept of gas exists to manage the network's resources efficiently, create incentives for nodes that perform computations, and provide a flexible pricing model.

The blocks in Ethereum are linked together similarly to Bitcoin in terms of their chain structure, with the linking process explained in Section 3.1. One difference between the blocks in these

two blockchains is the metadata they store. Ethereum blocks contain more metadata that can for example provide information about the state of the network. Just like each transaction receipt contains a bloom filter, each block also has a bloom filter. This filter enables clients to quickly search and check if a specific smart contract event occurred without having to process all the transactions stored in the block.

3.2.1 Smart contracts

A smart contract is a program that executes when certain predefined rules or conditions are met. This establishes trust and security while also eliminating the need for intermediaries. Smart contracts are a key feature of Ethereum and once deployed, smart contract code resides on-chain and cannot be changed or modified. However, developers do have techniques that can be used to upgrade the contracts. One technique for upgrading a smart contract is to transfer the state and functionality to another smart contract [22]. This process is typically performed through a script that deploys a new contract, copies all the necessary data to the new contract and updates all references in other contracts to point to the new contract.

An Ethereum smart contract is deployed by sending a transaction containing the compiled smart contract bytecode without specifying any recipient [22]. The bytecode is machine-readable code used by the EVM to execute the contract's code. To interact with a smart contract, you need to have access to the contract's Application Binary Interface (ABI). An ABI is a JSON representation of the functions, events, and other contents of the smart contract. The ABI defines how to interact with the smart contract and acts as a bridge between external applications and the contract's binary code [22]. Without a contract's ABI, external applications will not know how to encode and decode data sent and received from the smart contract.

Most Ethereum smart contracts are written in a programming language called Solidity [22]: a statically typed high-level language inspired by C++, Python, and JavaScript. Compiling a contract creates the smart contract's ABI and bytecode. In Solidity, smart contract events are named data structures which can be emitted during execution of a transaction. Events allow the contract to emit information to external applications that are not connected to the blockchain. The parameters in the data structure can be marked with the indexed keyword to make searching for them in transaction logs more efficient. Applications can set up an event listener that queries the logs and filters for specific events and indexed parameters [24]. Events can thus help broadcast information to other applications and be used to monitor and react to specific events in smart contracts.

Retrieving data from Solidity smart contracts can also be done through functions that are read-only, denoted view functions. These functions do not modify the state of the blockchain, meaning that they do not require a transaction to be executed and do not consume gas. Since no transaction is executed when reading data, an account is not necessary. This can be convenient for applications that only want to read data from the blockchain.

3.2.2 Decentralized Applications

Together, the EVM and smart contracts make it possible to create more advanced blockchain applications such as decentralized applications (DApp). The backend code of a DApp is the smart contract code which runs on the blockchain, rather than on a centralized server. This means that the blockchain serves as both a data storage mechanism and a platform for executing the application's logic [22]. DApps offer many benefits such as zero downtime, identity privacy, and complete data integrity. In addition, smart contracts are guaranteed to execute in predictable ways, which provides verifiable behavior. A few drawbacks are the performance overhead from all nodes having to validate every transaction and network congestion that can occur when the network is overwhelmed with transactions. Maintenance of DApps can also be difficult since a smart contract cannot easily be updated once deployed.

Web3.js is a popular JavaScript library which is often used for developing DApps. The library enables you to interact with a local or remote Ethereum node through HTTP, IPC or WebSocket [25]. Web3.js provides functionality for creating Ethereum accounts, generating cryptographic keys, and signing and sending transactions. In addition, it can be used to deploy and interact with smart contracts deployed on the Ethereum blockchain. Developers can use the library to create contract instances and call methods in the contracts. Event listeners can also be set up to listen for specific smart contract events. The library can be used in both server-side and browser-based applications.

3.2.3 Ethereum clients

There are several different Ethereum clients available that allow nodes to connect and join the Ethereum network. All these clients follow the same protocol specifications to ensure compatibility within the network. Geth is one of the original Ethereum implementations and is written in the Go programming language. Geth handles transactions, deployment and execution of smart contracts, contains an EVM, and supports various consensus mechanisms. To turn a machine into an Ethereum node Geth can be installed, configured, and run according to the

appropriate guidelines [26]. Every Ethereum client implements a JSON-RPC API, which is used by applications to interact with the Ethereum blockchain. JSON-RPC is a stateless Remote Procedure Call (RPC) protocol [22]. Libraries like Web3.js use this API to interact and communicate with the Ethereum clients.

The three different types of nodes an Ethereum client can run are full, lightweight, and archive nodes. A full node stores the full blockchain data, but periodically prunes the data, meaning it does not store state data back to the genesis block. The full node validates every block and transaction in the blockchain and provides data on request. A lightweight node only downloads the block headers and utilizes full nodes for other necessary information. These nodes do not participate in the consensus algorithm but can independently verify data they receive based on the state roots stored in the block headers. The benefits of lightweight nodes include not requiring powerful hardware and consuming less bandwidth. Additionally, a lightweight node can access the blockchain with the same security guarantees and functionality as a full node. An archive node stores everything a full node does and creates an archive of historical states. This type of node is necessary when someone wants to query data stored in old blocks. The drawback of archive nodes is that they are more resource-intensive since they typically store terabytes of data. Despite this, archive nodes are still incredibly useful for services like block explorers and chain analytics [22].

Regardless of the type of node (full, lightweight, or archive), all Ethereum nodes need to manage and store the network state. One of the key data structures that enable this is the Merkle Patricia Trie (MPT), which contains critical information about accounts, storage, smart contract code, and transaction receipts. The MPT data structure is a combination of a Merkle tree and a radix tree. A Merkle tree is a data structure that uses cryptography to enable efficient and secure verification of large data sets [27]. Radix trees are efficient and space-optimized data structures used for key-value storage [28].

Combining a Merkle tree and a radix tree enables efficient storage and retrieval of state data and provides cryptographic verifiability [22]. Note that the metadata in Ethereum blocks contain a state root, which is the hash of the MPT. Each time a node updates the state, the MPT is modified, and the resulting root hash is included in the block. This means that the network state is secured and cryptographically verified by the consensus algorithm. Ethereum clients store the block data and MPT on disk. Geth, which is a popular client, uses LevelDB to store its data. LevelDB is a key-value storage that has ordered mapping from string keys to string

values. The database was developed by Google and is optimized for high write and read performance [29].

3.3 GoQuorum

GoQuorum, also known as Quorum, is an open-source Ethereum client that can be used to run both private and permissioned networks. It is a lightweight fork of the Geth client and implements proof of authority consensus mechanisms [30]. GoQuorum shares many similarities with Geth like following the Ethereum protocol, supporting smart contracts, using an EVM, and it is written in the Go programming language. This means that core functionality like how transactions are processed, how smart contracts execute, how data is stored on disk, and interaction between nodes are the same in GoQuorum and Ethereum.

There are a few differences between GoQuorum and Ethereum since GoQuorum is designed for private and permissioned networks. The consensus algorithms are PoA oriented, and the available algorithms are IBFT, Clique, and Quorum Byzantine Fault Tolerance (QBFT). The P2P layer is changed so only nodes with permission can join the network. GoQuorum has added support for private transactions and private smart contracts, which are only visible to a specified group of participants. To enable both public and private transactions, the MPT has been split into a public MPT and a private MPT [30]. The private transactions and private smart contracts are handled by Tessera, which is a private transaction manager. Each of the GoQuorum clients has a Tessera component that runs alongside it. This means all regular nodes have a Tessera component that can encrypt, decrypt, and distribute private transactions to other nodes.

Although the block contents in GoQuorum and Ethereum are similar, the PoA consensus algorithms and private transactions result in block generation and block validation to be significantly different [30]. Another difference is that GoQuorum is by default configured to be a zero-gas network. This means that the pricing of gas is removed, but the concept of gas itself remains. Since GoQuorum is designed for private and consortium-based networks, there is no need to incentivize nodes using gas.

The two main types of nodes in GoQuorum are known as validators and regular nodes. Validators participate in the consensus process and are responsible for validating transactions and blocks. Regular nodes store a full copy of the blockchain and can send transactions that are handled by validators. The regular nodes provide the RPC interface for interacting with the blockchain and act as a gateway for applications that want to use the blockchain. There also

exists a third type of node known as a qlight node that uses its own client. These nodes are lightweight replicas of regular nodes, optimized to consume less resources. They store only a subset of the blockchain data but still allow users to access data. Qlight nodes rely on regular nodes to access data that they do not have a copy of.

3.3.1 Consensus Protocols

GoQuorum provides three different PoA consensus protocols to choose from when configuring a network. To ensure the network operates correctly all nodes must be configured with the same protocol. PoA consensus protocols can only be used when participants know each other and there is a certain degree of trust between them. Some of the important properties to consider when comparing consensus protocols are finality, speed, security, scalability, and the required minimum number of validators.

Finality in blockchains means that well-formed blocks cannot be reversed once added to the blockchain. In consensus protocols like PoW, the finality is probabilistic, and the deeper a block is in the blockchain, the higher the probability that the transaction cannot be reversed [31]. Immediate finality refers to the condition where a transaction or block is considered irreversible once it has been added to the blockchain. There is no possibility that a transaction or block can be changed or removed after it has been added to the chain. This assurance relies on the conditions necessary for consensus, such as network synchrony and a valid number of Byzantine nodes. The immediate finality property guarantees that the chain of blocks cannot experience a fork [32]. Forking occurs when the blockchain splits into multiple chains because the nodes have different views of the transaction history. Thus, a protocol that has immediate finality ensures that all nodes always agree on one consistent version of the chain.

The time it takes to reach consensus is of utmost importance as it determines the throughput of the network, affects scalability, and impacts overall performance. A fast consensus protocol improves the user experience for applications that use the blockchain. In addition, it can improve security by reducing the chance of forks and various attacks. A consensus protocol's resistance to attacks and other malicious behavior is also important to consider. A poorly designed protocol could weaken the integrity of the blockchain [33].

The consensus protocol used in a blockchain directly impacts the scalability of the network. How a protocol handles an increased number of nodes and transactions is important to consider. The performance of a good consensus protocol should not be ruined by an increase in nodes

and transactions. The choice of protocol can also determine the minimum number of required nodes to function correctly. Byzantine Fault Tolerant (BFT) protocols typically require that two-thirds of the validators are operating as intended [34, 35]. Note that Byzantine fault tolerance refers to the ability to continue functioning correctly and reach consensus despite nodes failing or sharing incorrect information to other nodes [32]. A protocol which is BFT requires at least 3f + 1 nodes, where f is the number of faulty nodes the system should be able to handle [36]. Therefore, 4 validators are the minimum number of nodes required for BFT consensus protocols to function correctly when up to one validator is unresponsive.

3.3.2 Istanbul Byzantine Fault Tolerance Consensus

IBFT is one of the protocols recommended by the developers of GoQuorum for production networks. The protocol uses a group of validators to determine if a proposed block should be added to the chain. Blocks are added in rounds and each round a validator is arbitrarily selected as the proposer. The proposer is responsible for constructing the block and sharing it with the other validators. If two-thirds of the validators agree on the validity of the block it is added to the chain. When the consensus round is over, a different validator is selected to be the proposer for the next block [37].

After a block has been appended to the chain by being approved by two-thirds of the validators, it cannot be changed. This means IBFT provides immediate block finality and ensures no transactions are changed after the validators agree to add a block. As the name implies, IBFT provides Byzantine fault tolerance. The protocol offers system stability if less than one-third of the validators are behaving incorrectly [37]. The GoQuorum developers do not recommend using IBFT with less than four validators. A network containing less than four validators can still produce blocks but cannot provide the finality guarantee [32].

The possible states for a validator in IBFT are awaiting proposal, preparing, ready, and round change. A validator in the awaiting proposal state is waiting to receive a block from the proposer. In the preparing state, a validator has received a proposed block, which it must validate and then notify other validators of the result. After doing this, the validator waits to receive messages from others. When a validator is in the ready state, it has validated the block and it exchanges commit messages with other validators. These messages indicate that they are ready to add the block to their local copy of the blockchain. Once a validator has received commit messages from at least two-thirds of the validators, it adds the block to the chain. The

round change state occurs when consensus is not reached within a given time limit or when a block fails to insert [37].

In GoQuorum, IBFT can be configured to manage validators through either block header selection or contract selection. With block header selection, validators propose and vote to add or remove validators using the JSON-RPC API. In contrast, contract selection relies on a smart contract to specify the set of validators [38]. Both methods for managing the validators are initially configured in the genesis file, which is used to configure the network when it is created.

In conclusion, IBFT in GoQuorum is a consensus protocol that offers immediate finality and Byzantine fault tolerance. It is designed for private networks and reduces the required infrastructure that other consensus algorithms like PoW require [37].

3.3.3 Docker and Quorum Developer Quickstart

Docker is a platform used to build, deploy, run, update, and manage containers. A container is a standardized, executable component that consists of application code and all the dependencies required to run the code in any environment. Containers can simplify development and delivery of distributed applications [39]. Docker compose is a tool that enables running multi-container Docker applications. Compose uses a YAML file to configure the application's services and allows developers to start the multi-container application with a single command [40].

One way of setting up a GoQuorum network is using Quorum Developer Quickstart (QDQ). This is a command-line tool that creates a local GoQuorum network. Each node runs as a Docker container and is managed by Docker Compose. QDQ also operates various other services within separate containers, such as block explorers, and monitoring tools like Grafana and Prometheus. Grafana provides real-time visual analytics and Prometheus is used as a time series database to store data logged by the nodes. By default, the network uses the IBFT consensus protocol and consists of four validator nodes and three regular nodes. Since each node operates as an isolated process in a Docker container, data storage is not shared between the nodes. Every container has its own file system where it stores the blockchain data.

3.4 Product tracing

Product tracing refers to the ability to track the journey of a product or a batch of products from its origin throughout all stages of the supply chain. These stages include the sourcing of raw materials, manufacturing, storage, and transportation. Traceability in the supply chain can help

ensure food safety, prevent fraud, verify the authenticity of products, and create consumer trust [11]. Modern supply chains can be highly complex and can consist of hundreds of stages [3]. This complexity makes it difficult to trace products using traditional supply chain management where each enterprise stores its data separately. Blockchain technology has the potential to enhance supply chain traceability through decentralized and transparent tracking of products using transactions. Product tracing systems that utilize blockchain can provide secure and tamper-proof logs that contain product histories.

Provenance data refers to metadata that records data origin and its processing history [41]. Collecting provenance data in each part of the supply chain can help create the detailed product history needed in product tracing systems. Another important aspect of product tracing is the usage of unique identifiers. Products with unique identifiers are easier to manage and track in the supply chain. Using international information systems and standards for unique identifiers can improve traceability, efficiency, and accuracy in the supply chain. The following subsections provide explanations of the unique identifiers used in SeaChain.

3.4.1 Food and Agriculture Organization

The Food and Agriculture Organization (FAO), a part of the United Nations, manages an information system called the Aquatic Sciences and Fisheries Information System (ASFIS). ASFIS contains a global reference list of aquatic species used for fishery purposes, which includes standardized names, taxonomic classification, and unique species identification codes for fish and other aquatic organisms. The primary aim of the list is to enable data exchange and communication among fishery organizations, researchers, and other stakeholders in the field [42]. The list for aquatic species is used in SeaChain to uniquely identify fish species.

The unique identifier for a species consists of three alphabetic characters and is known as a 3-alpha code. The three letters are usually assigned randomly but, in a few cases, they are related to the scientific or English name of the species. The ASFIS list of species currently contains 13417 different species. Since more than 17500 different 3-alpha codes can be generated using the 26 characters in the English alphabet, the database can be further expanded. The ASFIS list of species can only be updated by the Fisheries and Aquaculture Division of the FAO [42].

3.4.2 EAN-13

EAN-13 is a widely used 13-digit barcode standard managed by GS1, an international organization that maintains standards for supply chains. Using EAN-13, unique product codes

can be generated for products. An EAN-13 code consists of a country prefix, company prefix, product number, and check digit [43]. Table 3 explains the length and purpose of each part of the code. SeaChain uses EAN-13 product codes to uniquely identify products.

Country prefix	2-3 digits; Represents the country of the product's manufacturer.
Company prefix	4-7 digits; A unique identification code for the company.
Product number	3-6 digits; A unique reference number for a product.
Check digit	1 digit; A checksum based on the first 12 digits of the EAN-13 code.

Table 3: EAN-13 product code description

The country and company prefixes are both assigned by GS1. Large countries or those with more companies requiring unique identification codes can have a range of country codes instead of a single code. The length of the company prefix will vary depending on the number of products the company needs to identify. Each company is responsible for maintaining unique product numbers for their products. When added together, the country, company, and product number must always be 12 digits. The 13th digit, the checksum, is used for error detection. Figure 3 shows an example of what an EAN-13 barcode can look like. For instance, the country code 700 indicates that the company is in Norway and GS1 has assigned the company the unique code 12345. Since the product number is 0001 it can be interpreted as this being the company's first product. The checksum for the country code + company prefix + product number is 7.

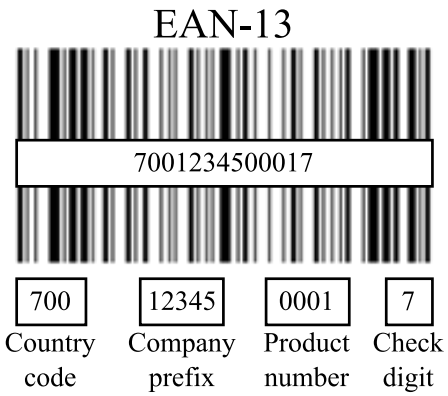

Figure 3: EAN-13 barcode

The checksum for an EAN-13 code is calculated using the 4-step algorithm displayed below [43]. In this algorithm, d_i represents the i-th digit of the first 12 digits in the EAN-13 code, where i ranges from 1 to 12.

1. $O = \sum_{i=1}^{6} d_{2i-1}$
2. $E = 3 * \sum_{i=1}^{6} d_{2i}$
3. $S = O + E$
4. $Checksum = \left(10 - (S \; mod \; 10)\right) mod \; 10$

In summary, product tracing is a crucial component of supply chain management that involves tracking a product throughout the supply chain. Implementing a product tracing system that employs international standards for unique identifiers, provenance data, and the inherent security and transparency provided by blockchain technology, presents a promising approach to improving supply chain management.

4 Design & Implementation

In this section we describe SeaChain: a smart contract-based product traceability system designed to track fish and related products throughout the supply chain in the Norwegian fishing industry. Utilizing a GoQuorum blockchain, SeaChain is built upon three distinct smart contracts, inspired by those proposed by Wang *et al. [3]* (see Section 2). Members of the GoQuorum network use SeaChain's API servers to deploy and interact with these smart contracts.

In SeaChain, fish and product information are added in batches and the system records their transfer history through the supply chain using the smart contracts. This creates an immutable chain of records on the blockchain that can be used to trace the origin of products. After a product has gone through the supply chain, consumers can scan a QR code that is on the product to view its transaction history.

Since the supply chain can be complex and involve many entities, this thesis considers only a simplified supply chain as displayed in Figure 4. First, a fishing vessel delivers fish to a fish factory. Then the fish factory produces a product and hands it over to a distributor, which ends up moving it to a retailer. Note that this simplification is only made to make our descriptions clearer. SeaChain is designed to handle any number of entities and is not limited to those depicted in Figure 4. To simulate a product moving through the supply chain, a script is used that sequentially interacts with the API servers.

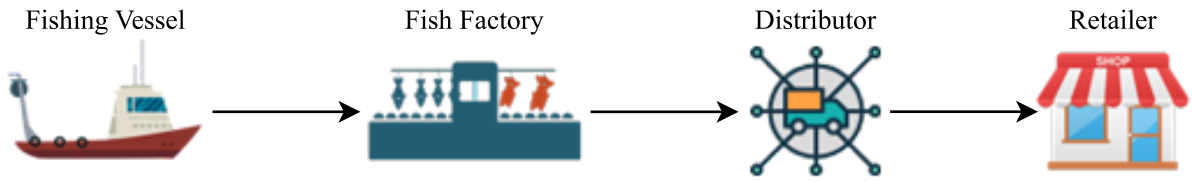

Figure 4: Simplified supply chain of the Norwegian fishing industry

The upcoming sections will first provide a detailed explanation of the GoQuorum network, the smart contracts, and the API servers. Finally, an overview of how the supply chain is simulated will be presented, illustrating how the components work together. All source code is available in the zip-file submitted with the thesis. Please refer to the included README file for a detailed explanation of the file structure and contents.

4.1 GoQuorum network

A local GoQuorum network has been set up using the Quorum Developer Quickstart tool. Each node in the network runs as a Docker container, and the validators use the IBFT consensus algorithm. Figure 5 provides an overview of the network, illustrating the various entities involved. The network consists of participants from the supply chain and the Directorate of Fisheries, a regulatory organization.

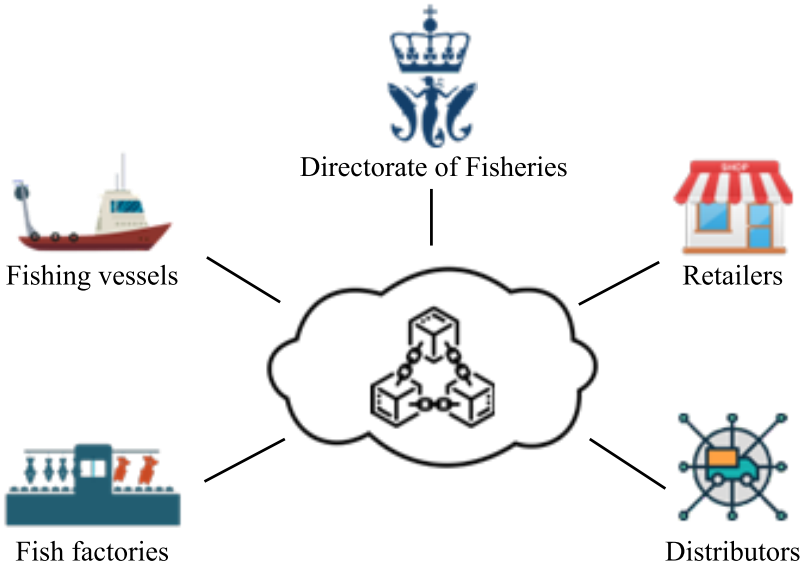

Figure 5: Overview of the participants in the GoQuorum network

Each entity controls a validator and regular node, except for the Directorate of Fisheries which has an additional regular node. This extra node is used to provide services such as querying product transaction histories. In total, the network consists of 21 nodes, including 10 validators and 11 regular nodes.

4.2 Smart contracts

The three smart contracts used in SeaChain are the Product Registration Contract (PRC), the Batch Addition Contract (BAC), and the Transaction Update Contract (TUC). The PRC is deployed once by the Directorate of Fisheries and is used to register products or fish species. A BAC is deployed for each registered product or fish species, providing the functionality to add a batch of that product or species. A TUC is deployed for each batch and contains the transaction history for the batch. These three smart contracts are designed to trace the fish and the products made with the fish separately. This means that the unprocessed fish and the product made with it have two separate transaction histories.

As previously mentioned, the contracts are inspired by those proposed by Wang *et al*. [3]. All three contracts have been refactored to a newer version of Solidity and updated with additional functionality. The most notable changes are the addition of events to improve communication with off-chain applications and design modifications that enhance the product tracing capabilities. These modifications include creating a link between the transaction history of the unprocessed fish and the final product, which enables traceability all the way back to the origin of the fish. Furthermore, the contracts track additional provenance related data such as weight and location. The structure and error handling of the smart contracts have also been significantly improved.

All SeaChain contracts contain authorization mechanisms that verify if the account initiating a transaction is permitted to interact with the smart contract. These access restrictions apply only to functions that add new data to the smart contract, without preventing other accounts from reading the stored data. Since GoQuorum is not a public network, access to smart contract data remains limited to those with network permissions. When a contract is deployed, the deploying account is granted administrator privileges, allowing them to manage authorizations for adding new data to the contract. Implementing these authorization mechanisms ensures that faulty applications or users with malicious intent cannot fill smart contracts with erroneous data.

Another similarity between the three contracts is the use of Solidity mappings, which are key-value data structures. Mappings enable efficient storage and organization of data in the smart contract, allowing fast searches using the keys. The contracts utilize these mappings for various purposes, including storing product, batch, and transaction data. Additionally, the mappings are used to provide various functions such as managing authorization and linking product codes to IDs.

The ABI and bytecode for each smart contract are required so applications can interact and deploy the contracts, respectively. Compiling the contracts to obtain the ABI and bytecode is achieved using a script. Most of the contracts are deployed by API servers. However, the PRC is only deployed once, making its deployment process unique. To reduce the possibility of multiple PRCs being deployed accidentally, a script has been written for its deployment. When this script is executed, the PRC contract is deployed by an account representing the Directorate of Fisheries.

4.2.1 Product Registration Contract

The data structure used by SeaChain to store both products and fish species, as well as the associated event, is illustrated in Figure 6. The data is stored in a mapping where the key is the number of registered products. Each product entry contains a product name, an EAN-13 product code, a 3-alpha code specifying the type of fish, an account address to the registrant, an address for the associated BAC, and a timestamp. Since fish species are registered using the same data structure as products, they contain the same information as a product. However, for a fish species, the product code is replaced by a 3-alpha code provided by the FAO.

```
struct Product {
    string productName;
    string productCode;
    string fishSpecies;
    address productOwner;
    uint timestamp;
    address BACAddress;
}
```

```
event ProductRegistered (
    uint indexed productId,
    string productName,
    string productCode,
    string fishSpecies,
    address indexed productOwner,
    uint timeStamp,
    address BACAddress );
```

Figure 6: Data structure and product event used in PRC.

The smart contract code presented in Figure 7 is used to register products. It requires four arguments, and the sender must be authorized to interact with the contract. The function starts with error handling to ensure that the product has not already been registered and that the provided arguments are valid. Next, the product is created using the provided arguments, the sender's address, and a timestamp from the blockchain. Subsequently, the smart contract updates multiple mappings with information about the new product. The total number of products is then increased, and the register product event is emitted. This event uses the indexed keyword to enable efficient transaction log filtering on product IDs and the product owner’s address. Since the code for registering a fish species is almost identical to registering a product, it is not shown here but can be viewed in the source code.

```
function productRegister(string memory productName, string memory productCode, string memory fishSpecies,
    address BACAddress) public reqAuthorized(msg.sender) {

  // Check if the product code has already been used
  require(bytes(productCodeToName[productCode]).length == 0, "Product code already registered.");

  // Check if the product name, product code and fish species meet the length requirements
  require(bytes(productName).length >= 3 && bytes(productName).length <= 64, "Invalid product name.");
  require(bytes(productCode).length == 13, "Invalid product code.");
  require(bytes(fishSpecies).length >= 3, "Invalid fish species.");

  // Check if the BAC address is valid
  require(BACAddress != address(0), "Invalid BAC address.");

  // Create a new product and update the product mappings
  products[numberOfProducts] = Product(productName, productCode, fishSpecies, msg.sender, block.timestamp,
    BACAddress);
  productCodeToName[productCode] = productName;
  productCodeToId[productCode] = numberOfProducts;
  productCodeToBACAddress[productCode] = BACAddress;

  // Increment the total number of products
  numberOfProducts++;

  // Emit the ProductRegistered event
  emit ProductRegistered(numberOfProducts - 1, productName, productCode, fishSpecies, msg.sender,
    block.timestamp, BACAddress);
}
```

Figure 7: Smart contract code for registering a product.

4.2.2 Batch Addition Contract

Batches of fish or products are stored in mappings using the data structure shown in Figure 8. Each batch stored in the contract contains a batch number, product code, account address of the sender, and the addresses of the related TUCs. Additionally, a batch stores a timestamp, weight, and GPS coordinates. This provenance data is used to help track the history and origin of products throughout the supply chain.

The fish and products are tracked separately in batches, and a TUC is deployed for each batch. To create a link between a batch of products and the batch of fish used to create the products, each batch contains a reference TUC. This reference is the address of the TUC belonging to the batch of fish used to create a product. By linking the batch of unprocessed fish and batch of products together in this manner, a complete product history is created. This history traces a product from the time the fish is caught until its delivery to the retailer.

```
struct Batch {
    string productBatch;
    string productCode;
    address batchManager;
    address TUCAddress;
    address TUCSupplyRef;
    uint addTime;
    uint weight;
    int32 latitude;
    int32 longitude;
}
```

```
event BatchAdded (
    uint indexed batchId,
    string productBatch,
    string productCode,
    address indexed batchManager,
    address TUCAddress,
    address TUCSupplyRef,
    uint addTime,
    uint weight,
    int32 latitude,
    int32 longitude );
```

Figure 8: Data structure and event used in BAC.

Adding a batch is done through the smart contract code presented in Figure 9. This function requires seven arguments and ensures that the sender is authorized. These arguments include the batch number, product code, TUC addresses, and other provenance data. The function first performs error handling to check that batch number has not been used before and that the provided arguments are valid. Since the Solidity programming language does not have native support for floats, the GPS decimal degrees are stored as integers using fixed-point arithmetic. The product batch is then created, and all batch related mappings are updated. Finally, the total number of batches is increased and the add batch event, defined in Figure 8, is emitted.

```
function addBatch(string memory productBatch, string memory productCode, address TUCAddress, uint weight,
      int32 latitude, int32 longitude, address TUCSupplyRef) public reqAuthorized(msg.sender) {

   // Check that the batch number has not already been used and that it is the correct length
   require(batchToAddress[productBatch] == address(0), "Batch has already been added");
   require(batchToReferenceAddress[productBatch] == address(0), "Batch has already been added");
   require(bytes(productBatch).length >= 18, "Batch number is too short.");

   // Check that the product code is the correct length
   require(bytes(productCode).length == 13 || bytes(productCode).length == 3,
      "Product code must be of length 13 or 3");

   // Check that the TUC address is valid and that it has not already been associated with a batch
   require(TUCAddress != address(0), "Invalid TUC address.");
   require(addressToBatch[TUCAddress] == 0, "TUC address has already been associated with a batch");

   // Using fixed-point arithmetic to check that GPS degrees are within valid range
   require(latitude >= -90 * 10000 && latitude <= 90 * 10000, "Latitude is not within valid range (-90 to 90)");
   require(longitude >= -180 * 10000 && longitude <= 180 * 10000,
      "Longitude is not within valid range (-180 to 180)");

   // Create a new batch and update the batch mappings
   batches[numberOfBatches] = Batch(productBatch, productCode, msg.sender, TUCAddress, TUCSupplyRef,
      block.timestamp, weight, latitude, longitude);
   batchToAddress[productBatch] = TUCAddress;
   batchToReferenceAddress[productBatch] = TUCSupplyRef;
   addressToBatch[TUCAddress] = numberOfBatches;

   // Increment the total number of batches
   numberOfBatches++;

   // Emit the BatchAdded event
   emit BatchAdded(numberOfBatches, productBatch, productCode, msg.sender, TUCAddress, TUCSupplyRef,
      block.timestamp, weight, latitude, longitude);
}
```

Figure 9: Smart contract code for adding a batch.

4.2.3 Transaction Update Contract

The transaction history of every batch is stored in TUCs using the data structure displayed in Figure 10. The data structure includes the hash of the current and previous transactions, the addresses of the sender and receiver, and a timestamp. The transaction hashes are 32-byte hexadecimal strings that uniquely identifies each transaction within the network. By using the number of transactions as the key in a mapping, an ordered list is created, providing the transaction history of a product.

```
struct Transaction {
    string currentTx;
    string previousTx;
    address sender;
    address receiver;
    uint time;
}
```

```
event TransactionAdded (
    uint indexed transactionId,
    string currentTx,
    string previousTx,
    address indexed sender,
    address indexed receiver,
    uint timeStamp );
```

Figure 10: Data structure and event used in TUC.

Updating the transaction history of a product is done using the smart contract code displayed in Figure 11. This function is called each time a batch of fish or products is transferred between entities in the supply chain. The function requires the sender to be authorized and takes the hashes of the current and previous transactions, as well as the address of the receiving entity, as arguments. Error handling is first performed to confirm the validity of the current and previous transactions and the receiver address. Then the new transaction is added to storage, the number of transactions is increased, and the add transaction event is emitted.

```
function addTransaction(string memory currentTx, string memory previousTx, address receiver)
        public reqAuthorized(msg.sender) {

    // Check that the current transaction and previous transaction have the correct format
    require(checkTransactionFormat(currentTx), "currentTx format is invalid.");
    require(checkTransactionFormat(previousTx), "previousTx format is invalid.");

    // Check if the receiver address is valid
    require(receiver != address(0), "Invalid account address.");

    // Create a new transaction and increase the total number of transactions
    transactions[numberOfTransactions] = Transaction(currentTx, previousTx, msg.sender, receiver,
        block.timestamp);
    numberOfTransactions++;

    // Emit the TransactionAdded event
    emit TransactionAdded(numberOfTransactions - 1, currentTx, previousTx, msg.sender, receiver,
        block.timestamp);
}
```

Figure 11: Smart contract code for updating the transaction history of a batch.

4.3 API servers

To interact with the smart contracts, each organization that participates in the GoQuorum network has an API server. In our simulated supply chain, the Directorate of Fisheries, fishing vessels, fish factories, and distributors utilize API servers. Retailers, however, do not need one as they are the final destination in the supply chain and do not update the transaction history of

products. The API servers are configured to use each entity's node credentials to communicate with the blockchain. A web3 instance, which enables interaction with the blockchain network, is created using each node's WebSocket JSON-RPC URL.

Each API server is built using JavaScript Express, a lightweight web application framework for Node.js. The framework simplifies the process of creating server-side applications and provides a robust set of features. These API servers use HTTP as the underlying communication protocol for exchanging data. Since each API server runs locally on the same server, they share a utilities file to reduce code duplication. This file contains elements such as the ABI and bytecode for each contract, as well as other shared functionality, like the code required for deploying smart contracts.

Event listeners are used by the API servers to automatically react to events emitted by the smart contracts. For instance, fishing vessels and fish factories listen for the species registered event to always have an up-to-date list of available species. Initially, the API servers communicated with the GoQuorum network using HTTPS. However, this was changed to WebSocket because the web3 event listener requires it. It is important to note that, although the API servers interact with the network using WebSocket, the endpoints for each API are accessed using HTTPS.

In our deployment scenario, the Directorate of Fisheries API server has many endpoints, and some of them are used by the other API servers. To simplify the explanation of this API, Table 4 provides an overview of each endpoint and a brief explanation. Since this API server belongs to a regulatory organization it has a list that contains the names and addresses of the different nodes in the network. It also has an overview of all nodes that are supposed to be able to add batches and register products. When this API starts up, it ensures all accounts who should have access to the PRC are authorized.

Endpoint	**Functionality**
GET request /prc-address	Used by other API servers to retrieve the PRC address. This ensures all entities use the official PRC and not a fabricated smart contract.
GET request /species-to-bac	Sends a key-value collection that contains 3-alpha species codes (keys) mapped to BAC addresses (values). This can be used by other API servers to easily obtain an overview of the different available species and their corresponding BACs.

GET request /species-to-name	Similar to the /species-to-bac endpoint but instead sends a key-value collection that maps 3-alpha species codes to their standardized name.
GET request /products	Retrieves all the registered products from the PRC and sends them as a list.
GET request /product-batches	Takes a BAC address or species/product code as query parameters. Sends a list of all the batches added for either a fish species or a product.
GET request /get-nodes	Sends a collection of contact information for the different nodes in the network. The contact information consists of the company name and their account address.
POST request /register-species	Used by the Directorate of Fisheries to register a new fish species in the PRC. A new BAC is created for the new species and all nodes in the network who should have access to it are authorized.

Table 4: API endpoints for Directorate of Fisheries

The API server that retrieves the product history of a batch, denoted the transaction history API, also belongs to the Directorate of Fisheries. This server has an endpoint that retrieves the complete transaction history of a product. The endpoint requires the product code and batch number as query parameters. The product code and batch number are used to retrieve the TUC address of the product batch and the fish batch used to create the product. Then all the recorded transactions for both the fish batch and the product batch are retrieved from the smart contract in separate lists. After this, the previous transaction field in the first transaction of the product batch is updated to reference the last transaction of the fish batch. The process of linking the fish batch and product batch is illustrated in Figure 12. Formatting is then performed on the data to present it in human-readable format. A few examples include converting Unix time to a regular date format, changing account addresses to company names, and dividing GPS decimal degrees due to fixed-point arithmetic. Finally, an HTML template is rendered using a templating engine to display the transaction history to a consumer. When a consumer scans the QR code on a product, this is the API endpoint they are taken to.

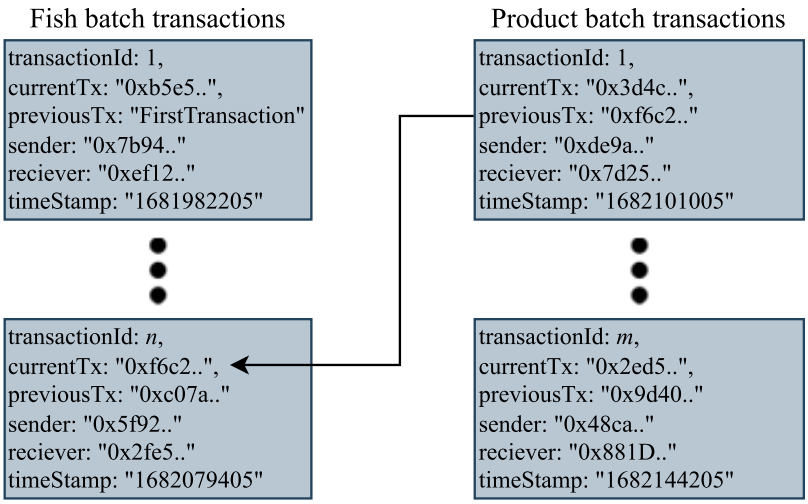

Figure 12: Linking fish batch and product batch transaction histories.

The API servers for fishing vessels, fish factories, and distributors all have an endpoint that enables them to update the transaction history of a batch. Updating the transaction history occurs after the product is transferred from one entity to another, making the address of the receiving entity necessary. Additionally, the species code or product code, along with the batch number, is required to retrieve the address of the TUC belonging to the batch. The total number of transactions is then read from the TUC and used to determine if this is the first transaction recorded. This information is necessary for setting the previous transaction field in the data structure of the new transaction. For the first transaction, the previous transaction field is set to the string "FirstTransaction". In all other cases, the field is set to the hash of the previous transaction. After this, the new transaction is created, and the smart contract function to add a transaction is called to store it permanently. To help illustrate how updating the transaction history occurs, Figure 13 displays a flow chart of the process. Note that the dashed lines in the flow chart indicate that it is not possible to loop back.

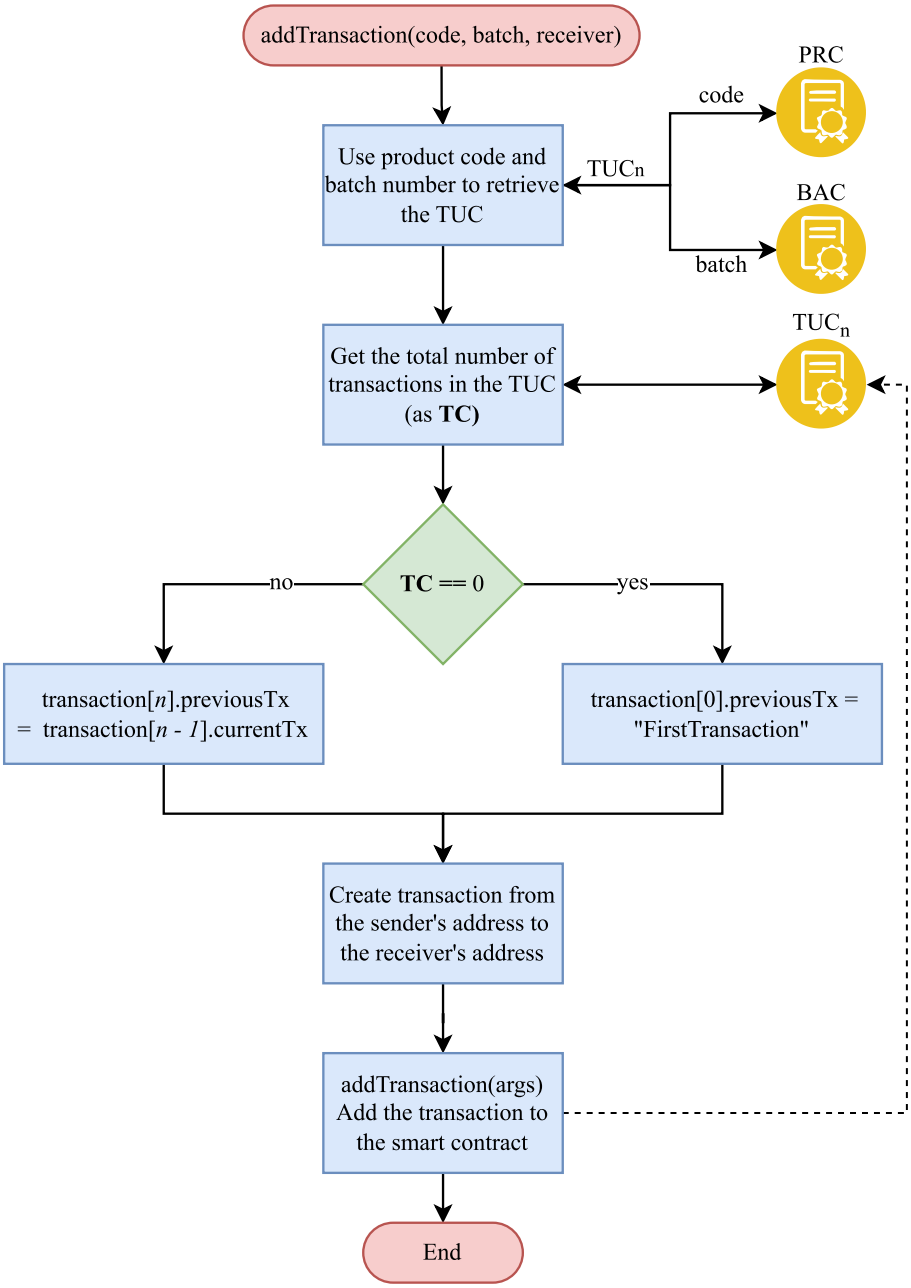

Figure 13: Flow chart for updating the batch transaction history.

Fishing vessels and fish factories both have a similar endpoint for adding batches. Note that we assume the fishing vessel API servers automatically register batches based on data received from the system that automatically collects fish data (see section 1.4). To add a fish or product batch, the API servers deploy a new TUC, authorize the appropriate accounts, generate a unique batch number, and call the add batch function in the BAC. Unique batch numbers are generated by using the date (DD.MM.YY), time (HH:mm), and the variable in each BAC that keeps track of the total number of batches (Total Count or TC). The format of the batch number is "DD.MM.YY-HH:mm-TC".

A major difference between adding a fish batch and a product batch lies in the links they establish. While a product batch is linked to the fish batch used to create the product, a fish batch has no such connection. This is because fish is viewed as a raw material in a product rather than a product itself. For example, consider fish batch A and product batch B, where batch B is made using batch A. Batch A sets the TUC supply reference (see data structure in Figure 8) to the zero address, commonly used to indicate the absence of an address. In contrast, product batch B sets the TUC supply reference to the TUC address of batch A. This is done to create a complete product history, as mentioned in Section 4.2.2.

The fish factory API server also features an endpoint for registering a new product. To register a product, a product name, a species code for the fish used, a country code, and company code are required. First, a BAC is deployed for the new product, followed by retrieving the number of registered products from the PRC. The country code, company code, and number of products are used to generate a unique EAN-13 product code. Once the product code is generated, the register product function in the PRC is called to create the product.

4.4 Simulating the supply chain

Since SeaChain has not yet been deployed in a product setting, the supply chain and product data must be simulated. A script has been written for this purpose and it sequentially interacts with the API servers to portray the supply chain. There is no mechanism to verify the delivery of a batch. Therefore, we assume that each entity receives their batch of fish or products when it is transferred in the supply chain, and the entity transferring the batch is notified upon successful delivery.

Figure 14 displays a flow chart of the supply chain script, illustrating how the API servers interact with the smart contracts. The figure also contains a legend with color codes that represent which entity an API server belongs to. Note that the dashed lines in the flow chart indicate that it is not possible to loop back. The script follows these steps:

1. The Directorate of Fisheries registers a new fish species using the PRC.
2. A fishing vessel adds a new fish batch by interacting with a BAC.
3. The fishing vessel transfers the batch to the fish factory, then it updates the transaction history in the TUC for this batch.
4. The fish factory registers a new type of product in the PRC and adds a new product batch using the fish batch it received.

5. The product batch is transferred to a distributor, and then the fish factory updates the product batch transaction history.
6. The distributor transfers the product batch to a retailer and then updates the transaction history.
7. Finally, the script generates a QR code that can be scanned to view the transaction history.

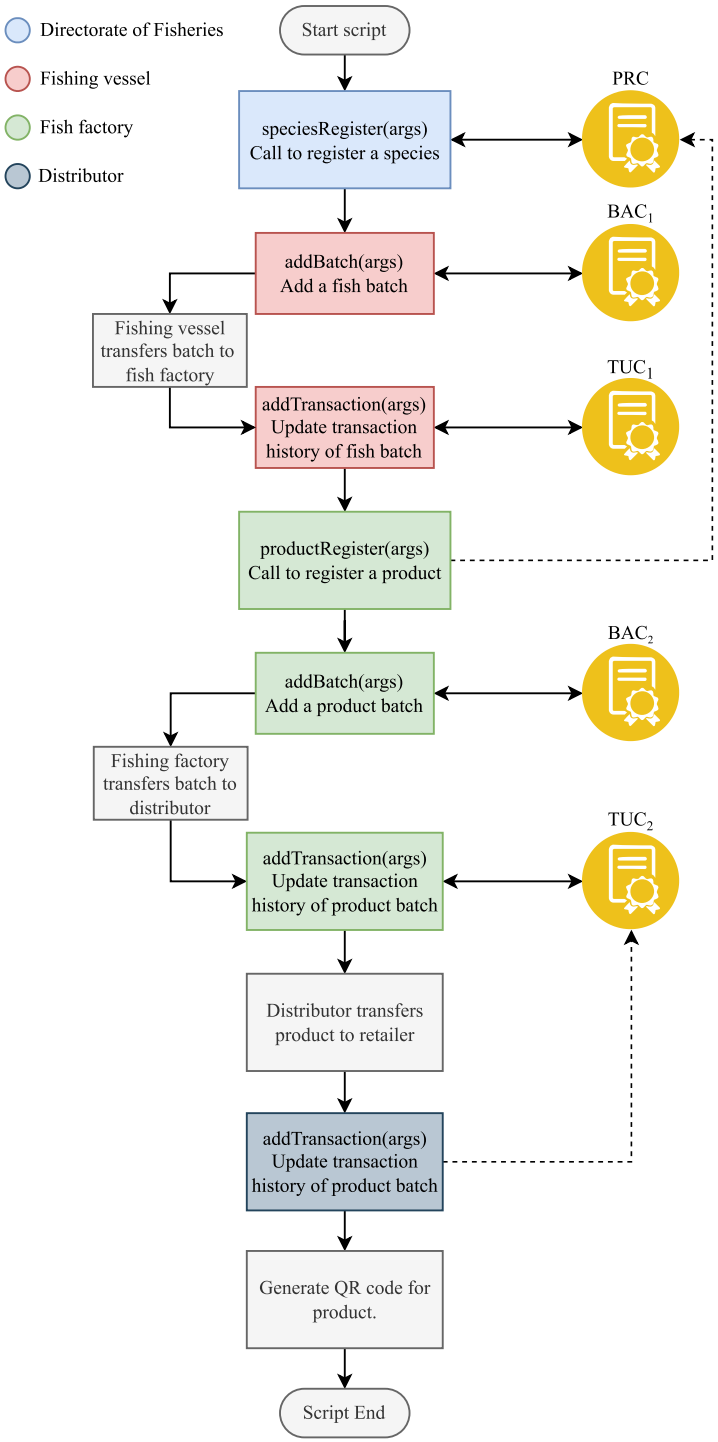

Figure 14: Flow chart for the supply chain simulation script.

By scanning the QR code generated by the supply chain script, consumers can access important information about the product. The transaction history API retrieves product information, origin of the fish, and the transaction history. The process is displayed in Figure 15, and the product history is linked together as shown in Figure 12.

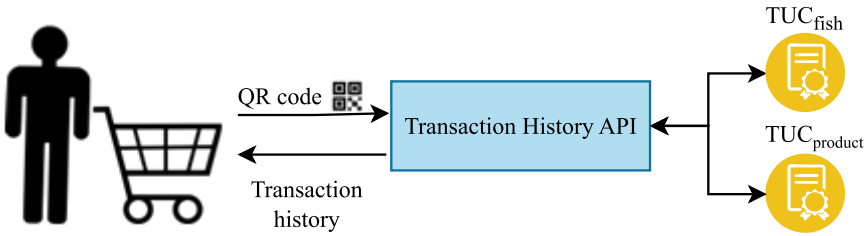

Figure 15: Accessing the transaction history.

The data displayed to consumers when accessing the webpage on PC can be viewed in Figure 16. The product history table tracks the journey of fish and product batches throughout the supply chain, with the sender and receiver fields indicating the addresses of various entities involved. To make the information more easily understandable for consumers, the addresses are converted into company names. The time field denotes the moment when a batch is transferred from the sender to the receiver. The previous and current transaction fields display unique transaction hashes originating from the blockchain, serving as evidence for the transaction history's validity. It should be noted that since this thesis utilizes a simplified supply chain model, the product history is quite short. If SeaChain were used in a production environment, each batch would have a more extensive history.

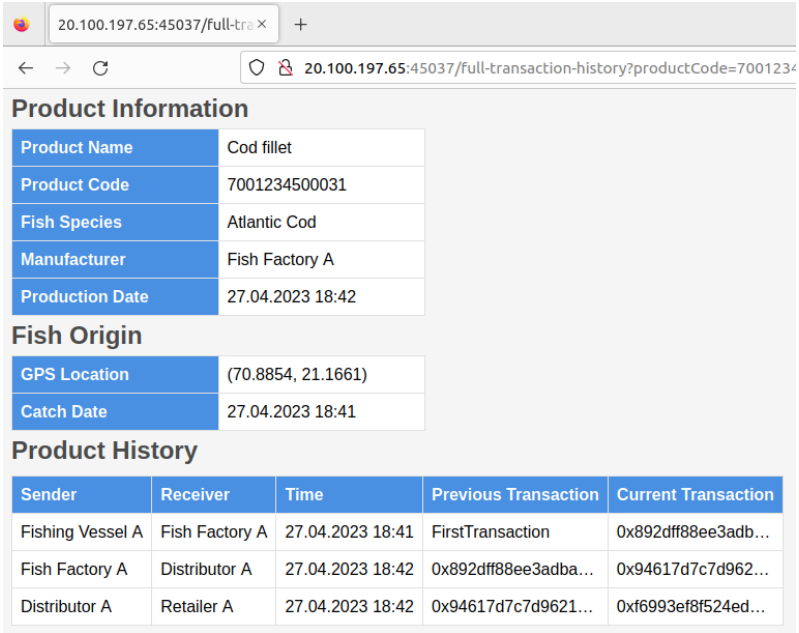

Figure 16: PC screenshot displaying the page consumers view by scanning QR codes.

Since the product history is accessed by scanning a QR code, the page has been designed for mobile compatibility as well. Figure 17 illustrates the mobile layout, with the primary difference being tables optimized for screen size and the addition of horizontal scrolling functionality added to the product history table.

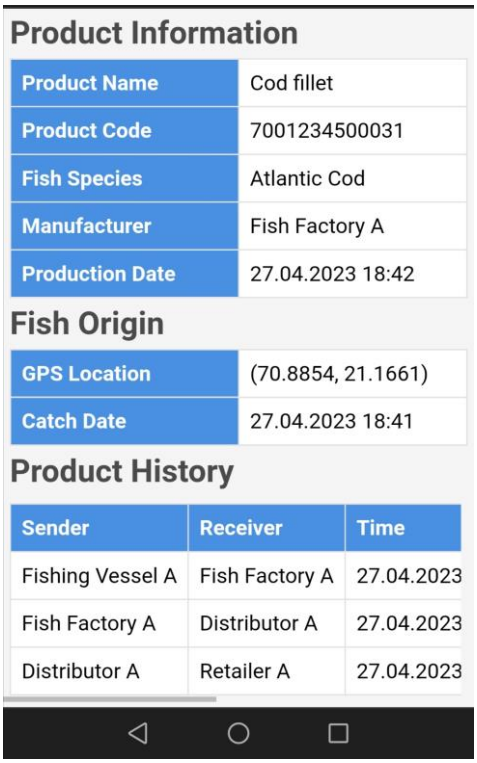

Figure 17: Smartphone screenshot displaying the page consumers view by scanning QR codes.

5 Evaluation

This section evaluates the performance of the three smart contracts, the core functionality of the API servers, and the GoQuorum network. Table 5 outlines the experiments conducted to analyze performance, the metrics collected, and their respective purpose. These metrics have been selected since they are standard benchmarks for blockchain networks and other systems. In this evaluation, transaction throughput refers exclusively to the network's capacity to process smart contract transactions. This emphasis aligns with the thesis's relevance and objectives, as the measurement of regular transactions, such as cryptocurrency transfers, is not considered necessary for this analysis. Latency provides insights into the delays involved in transaction processing and the responsiveness of each API endpoint. Additionally, hardware utilization offers an understanding of the system's resource efficiency. By examining these metrics under varying workloads, we can measure system performance and identify potential bottlenecks.

Experiment	**Metrics**	**Purpose**
Evaluation of smart contracts with various numbers of validators	Transactions per second, transaction latency	Measure the performance and efficiency of each smart contract and see how the network scales with more validators
Evaluation of API endpoints for registering products, adding batches, and updating transaction histories	Response latency	Determine the responsiveness of the API servers
Product history retrieval	Latency for retrieving product histories	Assess the efficiency of product history retrieval and API server performance
GoQuorum network performance	CPU usage, memory usage, network traffic, disk usage, database usage	Evaluate the resource utilization of the GoQuorum network

Table 5: Overview of the experiments conducted, the metrics collected, and their purpose.

All metrics, except for those related to hardware utilization, have been collected using scripts that run multiple benchmarks. This method provides a more accurate representation of

performance, allowing for the calculation of standard deviations to better understand the variability of results. The benchmarking scripts are included in the source code submitted with the thesis. It is important to note that the GoQuorum network and API servers used in this thesis are running in a centralized manner on an Azure virtual machine (VM). While decentralizing the network was beyond the scope of this thesis, such a setup could potentially yield different experimental results. The hardware specifications for the VM are detailed in Table 6.

Component	**Specification**
Processor	Intel(R) Xeon(R) Platinum 8171M (8 cores available)
Memory	32 GiB
Storage	64 GiB SSD (Max 12800 IOPS, 192 MBps)
Network	Max 4 NICs. Expected network bandwidth: 4000Mbps
Operating System	Ubuntu 20.04 (Focal Fossa)

Table 6: Hardware specifications for the server used to benchmark the system.

Hardware utilization data was collected from each node in the network using Prometheus, a monitoring tool that functions as a time series database. The collected data was visualized with Grafana, an analytics and monitoring tool that has integration support for Prometheus. Data was logged under both idle and heavy workload conditions to capture the network's operation under different scenarios. Queries to the Prometheus database were specifically configured to select the average performance of validator and regular nodes, respectively. To simulate heavy workloads, a script testing the transaction throughput was executed during the performance measurement for both validator and regular nodes.

The details of each experiment, including the steps taken during each test, will be discussed in the upcoming section. The section will also include an analysis and detailed discussion of the results for each experiment.

5.1 Results

The Transactions Per Second (TPS) metric was measured by running a script designed to register products, add batches, and update transaction histories. The script sent batches of concurrent transactions to the network nodes, distributing the transactions evenly to avoid potential bottlenecks. To focus on the pure transactional performance of the smart contracts, the transactions were sent directly to the contracts, circumventing the API servers, which add

an extra layer of processing. The experiments were performed for each smart contract with networks containing 4, 10, and 20 validators. The results of these benchmarks are displayed in Figure 18, Figure 19, and Figure 20.

The results suggest that the TPS can vary significantly, which is likely due to competition for hardware resources. The upcoming hardware utilization analysis confirms that CPU resources were indeed a point of contention during these benchmarks. As the number of validators increased, each node had access to a smaller share of the CPU resources. Keeping this in mind while analyzing the graphs, it becomes evident that the difference in TPS between networks with 4 and 20 validators is not as substantial as one might initially expect.

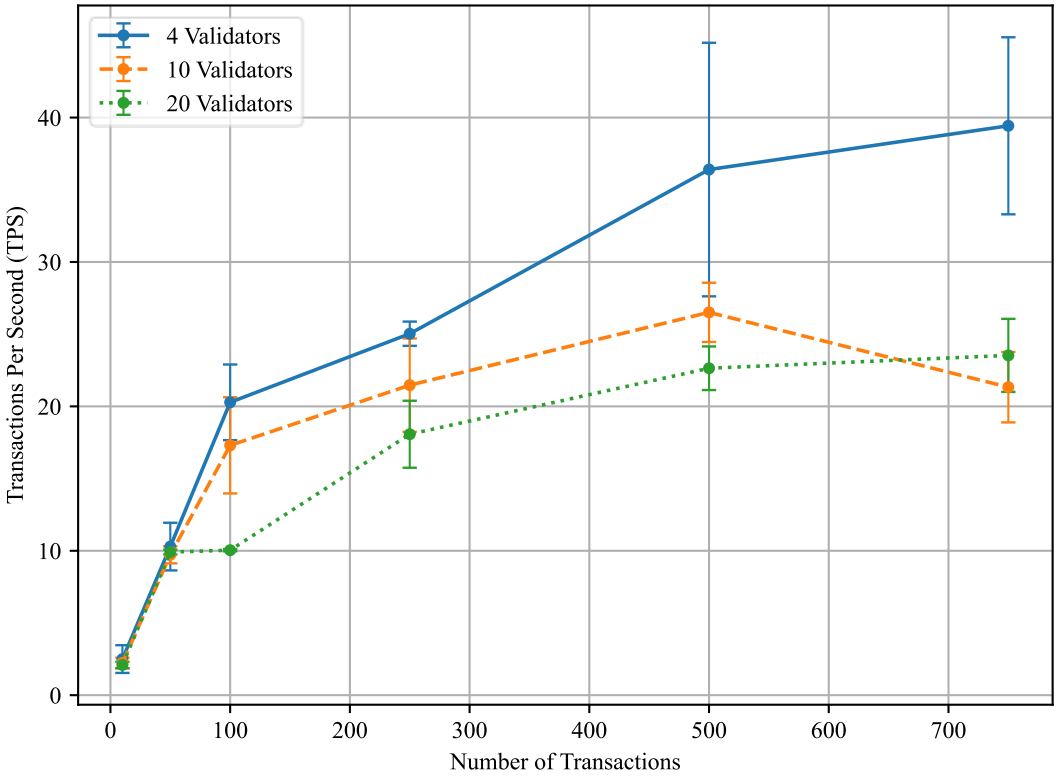

Figure 18: Transaction throughput when registering products in the PRC for different number of validators under varying workloads.

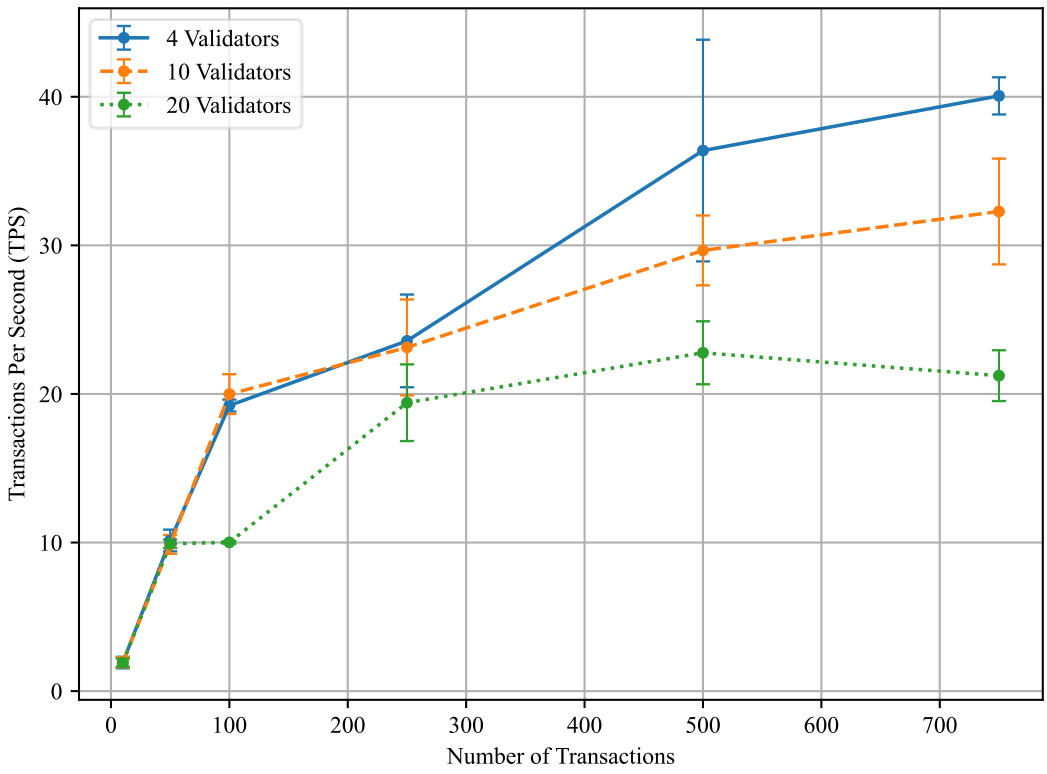

Figure 19: Transaction throughput when adding batches in the BAC for different number of validators under varying workloads.

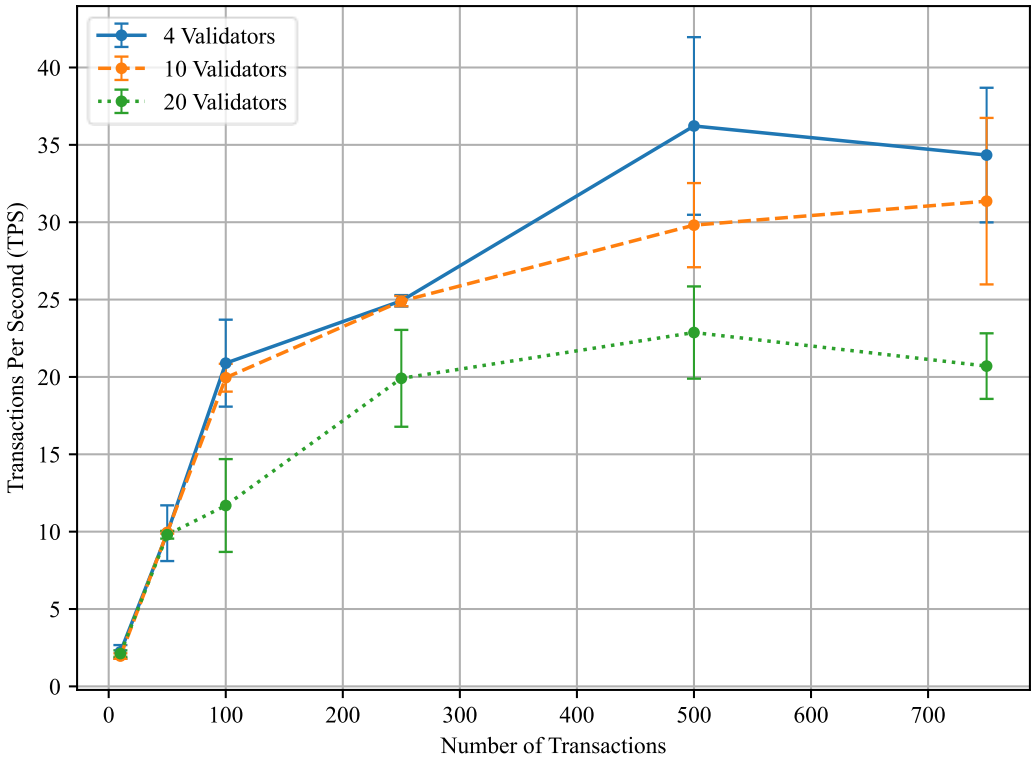

Figure 20: Transaction throughput when updating transaction histories in the TUC for different number of validators under varying workloads.

The transaction latency, defined as the duration from the submission of a transaction until it is confirmed to be added to the blockchain, was measured under varying workloads to evaluate scalability under network constraints. The results of this experiment, illustrated in Figure 21, suggest a relatively linear relationship between the workload and the latency. However, the standard deviation indicates an increased variability in latency as workloads increase, possibly due to limited hardware resources.

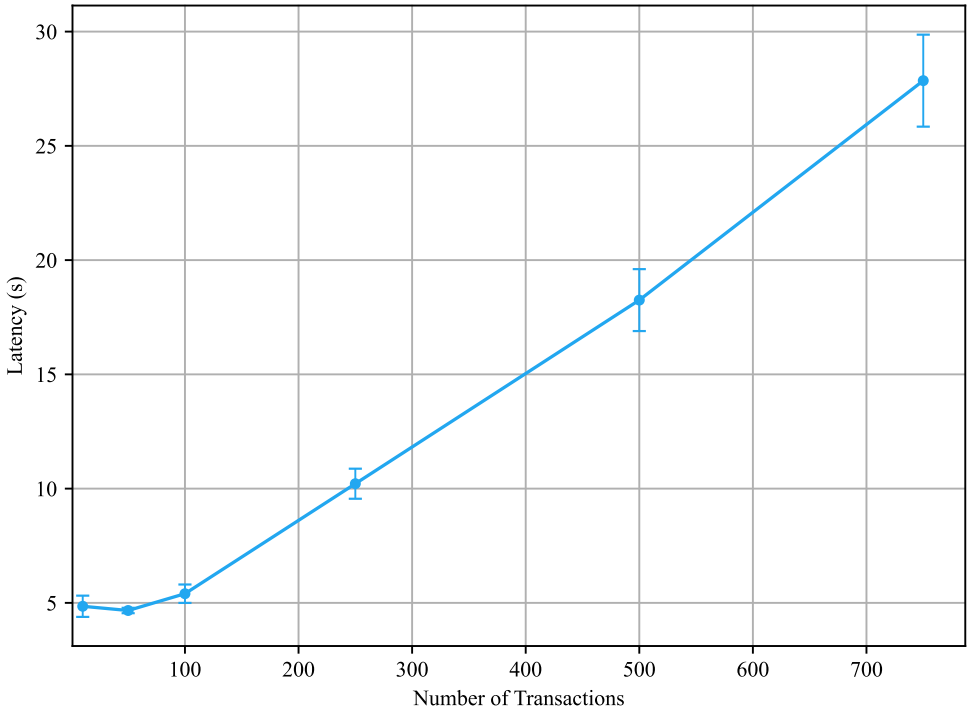

Figure 21: The average latency for smart contract transactions for various workloads.

The core functionality of the API servers utilized in the supply chain includes registering products, adding batches, and updating transaction histories. Therefore, the latency associated with these API interactions was measured. Furthermore, the duration required to simulate the entire supply chain – from the moment a batch of fish is caught until a product is delivered to a retailer – was also assessed. The results of this experiment are presented in Table 7. The average response time is significantly higher than the TPS achieved when directly interacting with the smart contracts. This difference arises because the API servers are tasked with essential operations such as deploying new contracts, processing data, and performing error handling.

Functionality	Average latency (s)
Register product	9.9
Add batch	15.2
Update transaction history	10.3
Supply chain simulation	84.7

Table 7: Response time for API endpoints used to register products, add batches, update transaction histories, as well as the time required to run the supply chain simulation script.

Consumer retrieval of product histories through an API server was evaluated for various workloads. The experiment measured the average latency of the API server in fetching and processing product histories in response to batches of concurrent calls to the API endpoint. The results, displayed in Figure 22, show a linear performance trend, which is expected given the concurrent call handling capability of the Express framework used in creating the API servers. The average latency for retrieving a single product's history is approximately 35ms, a delay so brief it would likely be unapparent to a consumer.

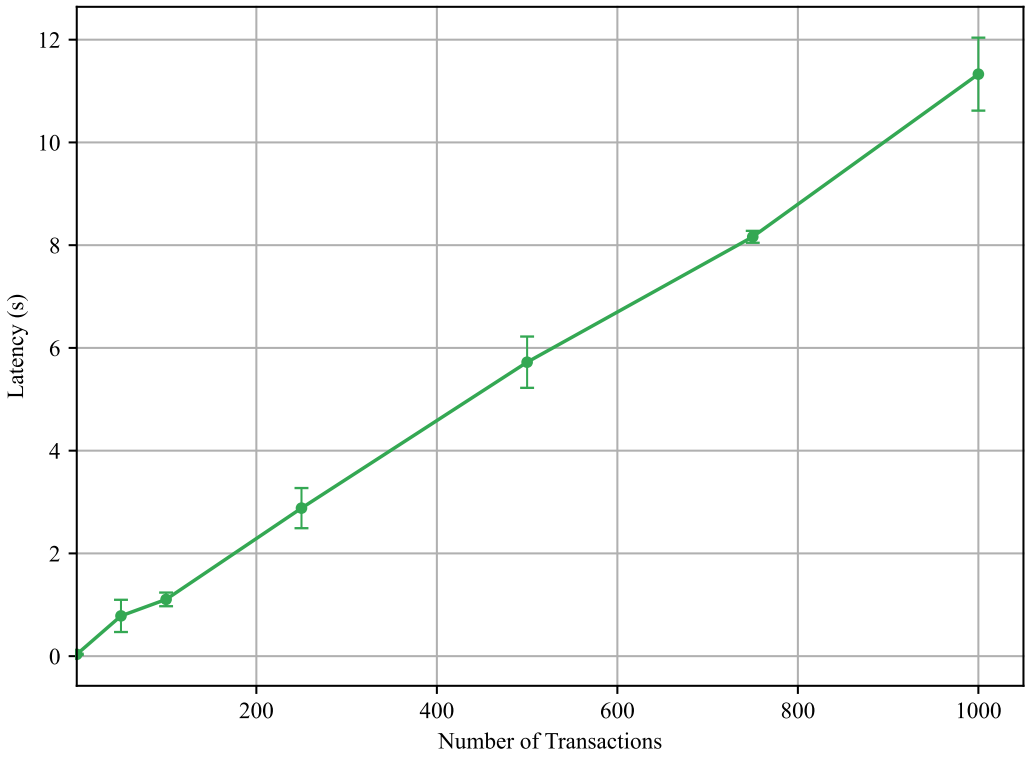

Figure 22: Average API server latency for retrieving product histories for various workloads.

Hardware utilization for both validators and regular nodes was benchmarked using the same methodology. This involved first measuring a few minutes of idle load, followed by running

the TPS benchmark that sends concurrent transactions, and finally capturing a few more minutes of load to observe any residual effects. Notably, the script sending these concurrent transactions incorporates a brief pause between each batch to prepare the next set of transactions. This pause is visible in certain graphs and will be further clarified within the discussions accompanying any graph where its influence on the visual representation is significant.

Figure 23 presents a time series line chart illustrating the CPU utilization during idle workload and the TPS benchmark execution. Given the use of a multi-core VM with eight cores, the maximum CPU utilization for the VM is represented as 800%. The results reveal that during idle load, the CPU uses approximately 1/8 of its total capacity. Conversely, during the benchmarks, it operates at maximum capacity while processing the concurrent batches of transactions sent by the script. This suggests that the system performance may be constrained by the CPU, and more powerful hardware could potentially enhance performance. The substantial dips evident in the graph occur when the validators have completed processing the transactions and the script is preparing to send the next batch of concurrent transactions.

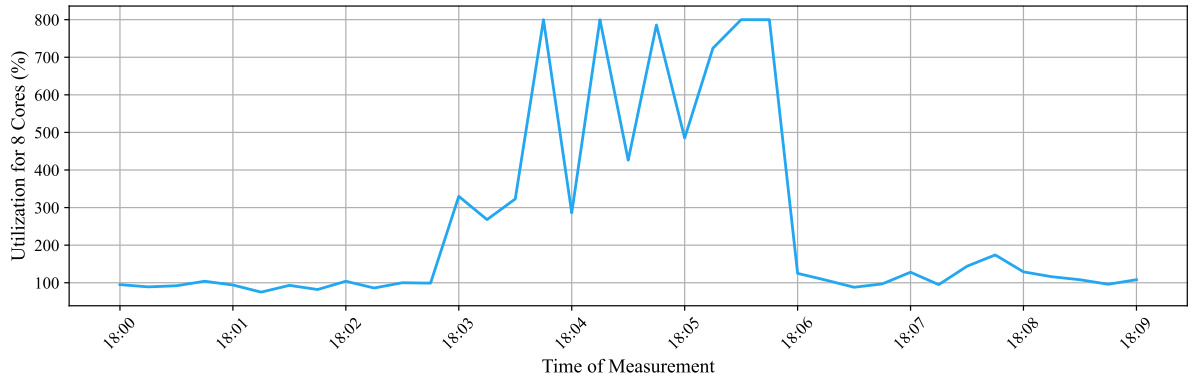

Figure 23: A time series line chart displaying the servers CPU utilization during idle and heavy workloads.

The average memory utilization by validators and regular nodes is illustrated in Figure 24. This graph tracks both the memory actively used by each node and the memory reserved by them. During benchmarking, a substantial surge in memory usage was observed for both validators and regular nodes. Interestingly, the memory reserved is much greater than what is currently being utilized, likely because nodes allocate memory in anticipation of future needs and do not instantly deallocate memory once tasks are completed. The data suggests that validator nodes consume slightly more memory during idle load, while regular nodes are more memory-intensive during heavy workloads. The larger memory consumption by regular nodes during heavy workloads likely stems from the benchmarking script distributing concurrent transactions

only to regular nodes, resulting in significant memory usage. During some experiments, the memory limit of the VM was reached, which constrained the experiments to sending no more than 750 concurrent transactions.

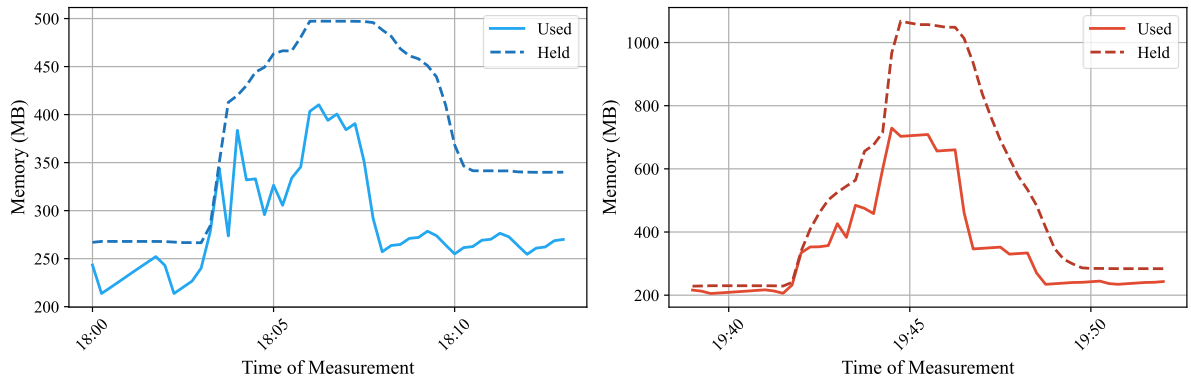

Figure 24: The average memory utilization by validators (left) and regular nodes (right) during idle and heavy workloads.

Figure 25 illustrates the average network traffic for both validators and regular nodes, capturing both ingress (incoming) and egress (outgoing) network traffic. The data demonstrates that validators consistently transmit and receive more data than regular nodes, a trend observable under both idle and heavy load conditions. It is worth noting that during idle load, the ingress and egress network traffic of validators is approximately five times higher than that of regular nodes. This is expected, given that validators must engage in extensive communication to reach consensus on the state of the blockchain.

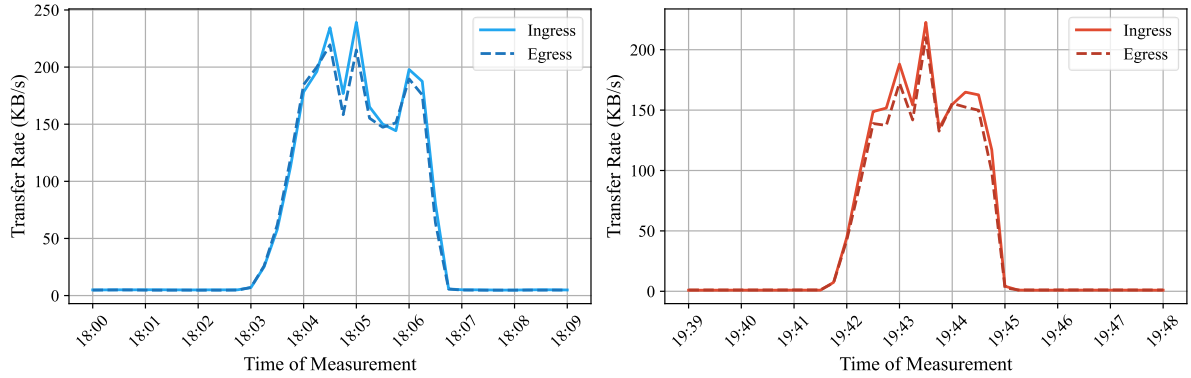

Figure 25: The average network traffic for validators (left) and regular nodes (right) during idle and heavy workloads.

The rate at which data is read and written to the LevelDB database (which stores the chain data) by each node in the network is illustrated in Figure 26. Since there were no requests for data during the idle load or during the benchmarking, the read rate is consistently zero in the graphs.

Both validators and regular nodes have a similar write speed to the database during idle and heavy workloads. This makes sense since regardless of node type, the chain data stored in the LevelDB database should be the same.

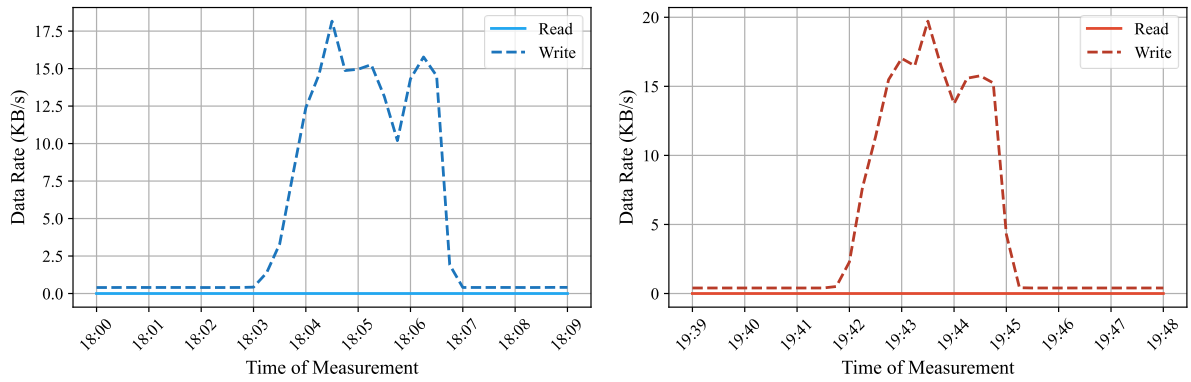

Figure 26: The average LevelDB I/O for validators (left) and regular nodes (right) during idle and heavy workloads.

The average read and write rate for disk activity among validators and regular nodes are illustrated in Figure 27. The graph reveals that validators have a higher I/O rate than regular nodes. Although the scale of the graph makes it less obvious, the idle write rate for validators is double that of regular nodes, and the read rate nearly quadruples. Comparing this to the average LevelDB I/O portrayed in Figure 26, the disk usage is notably higher. This difference can be attributed to the fact that the disk utilization data considers all disk access operations, including logging, Docker I/O, and other necessary disk activities.

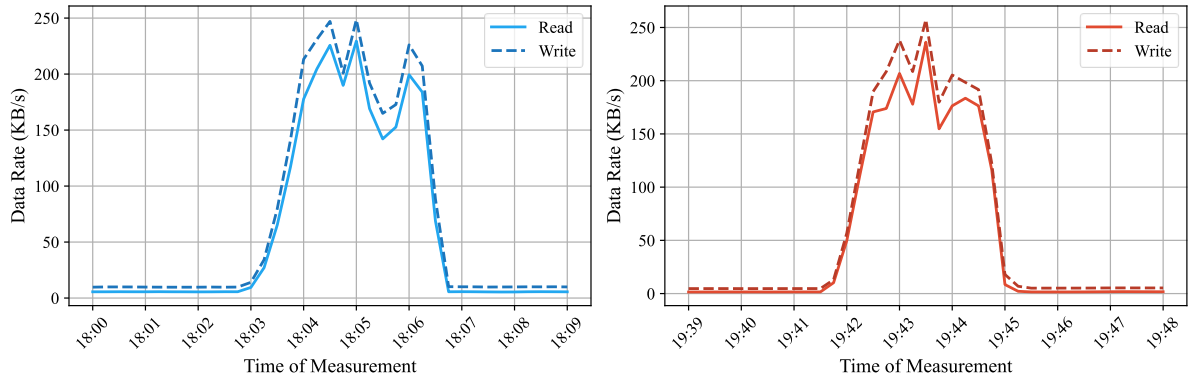

Figure 27: The average disk I/O for validators (left) and regular nodes (right) during idle and heavy workloads.

In summary, the experiments demonstrate that the GoQuorum network possesses solid transaction throughput and low latency. The smart contracts operate efficiently, and despite hardware constraints limiting the extent of testing, the network displays potential for scaling

with an increase in validators. The API servers that register products, add batches, and update transaction histories provide responses within a reasonable timeframe, considering the tasks they perform. The retrieval of product histories experiences minimal delay, and the API server displays linear performance under concurrent workloads. Hardware benchmarks indicate that the system's performance is bottlenecked by the CPU utilization and the 32 GiB memory limit. However, these limitations stem from running the network locally, suggesting that a decentralized network will likely not encounter the same issues with CPU and memory.

6 Discussion

This section is divided into three distinct parts, each providing deeper insight into different aspects of our research. We begin with a security analysis of SeaChain, with the aim of assessing its robustness and identifying potential vulnerabilities. The second part involves a comparison of SeaChain with the existing product tracing systems previously discussed in Section 2. In the final part of this section, we reflect on the achievements of our research, evaluating if the objectives have been met and to what extent our findings support the thesis statement.

6.1 Security analysis

Ensuring the confidentiality, integrity, and availability of supply chain data is crucial to be able to enforce regulatory compliance and provide consumers with legitimate information. In this section, we perform a security analysis of SeaChain. Threats are identified and mitigation strategies are proposed. Although the GoQuorum network utilized in this thesis is operating locally, this analysis will consider a fully decentralized deployment as intended in a practical application. The objective is to analyze the potential vulnerabilities, evaluate SeaChain's robustness against these threats, and suggest areas for further improvement.

The identified threats, along with their description and proposed mitigation strategies, are presented in Table 8. This table serves as an overview of the identified threats, and the remaining part of this section will provide a more in-depth review of each threat.

Threat	**Description**	**Mitigation Strategies**
Unauthorized access	Unauthorized entities gaining access to the system	Implement robust authentication and authorization mechanisms
Supply chain data integrity threats	False or inaccurate data is input into the system	Deploy systems that automatically register data in a trusted and reliable manner.
Smart contract vulnerabilities	Bugs or vulnerabilities in smart contracts that can be exploited to manipulate the behavior of the system	Regularly conduct security audits. Use formal verification methods to ensure that a smart contract's logic meets predefined specifications

Denial of service (DoS) attacks	Overwhelming the API servers with traffic, rendering services unavailable	Employ rate limiting and DoS protection services
Insider threats	Organizations with legitimate access to the system misuse it with malicious intent	Regular government-controlled audits. Establish clear sanctions for misuse
Physical security threats	Threats to the physical infrastructure hosting the API servers and nodes in the network	Secure physical infrastructure locations. Employ hard-drive encryption.

Table 8: Overview of identified threats and the proposed mitigation strategies

Despite the assumption of implementing secure communication standards, such as TLS/HTTPS, on the API servers, unauthorized access to the system remains a significant threat. An attacker who has successfully obtained valid credentials might submit fraudulent transactions or leak sensitive information. To mitigate this risk, it is crucial to implement robust authentication and authorization mechanisms, such as secure passwords, two-factor authentication, API keys, and regular access reviews. In addition, logging system activities and continuous monitoring for any unusual behavior are essential steps for enabling swift responses in case of a security breach.

The challenge of ensuring data integrity in the supply chain is of utmost importance in a product tracing system. False or inaccurate data compromises the reliability of the system, which could lead to loss of trust and illegal activities going unnoticed. SeaChain relies on fishing vessels being equipped with systems that automatically register data regarding the fish caught. This automated system avoids the possibility of manual input errors and makes it difficult for false data to be entered maliciously. However, even with automated data entry, errors can still occur. Therefore, incorporating data validation mechanisms at various stages of data capture and entry is essential. These mechanisms can help identify and correct errors, ensuring that only accurate and reliable data is written to the blockchain.

Smart contracts are quite different from regular applications that can be patched whenever a vulnerability or bug is found. Although there are methods for updating smart contracts, they are not as straightforward as simply changing the code that causes the vulnerability. If vulnerabilities are not detected before deployment, smart contracts can be exploited, putting the

system's integrity at risk. Due to these challenges, it is crucial to keep the development of smart contracts simple and clearly document them to minimize bugs. Furthermore, thorough testing and regular security audits should be conducted before deploying the smart contracts. To provide stronger guarantees of functional correctness, formal methods can also be used in the development and verification process.

The API servers in SeaChain serve as crucial gateways for interacting with the smart contracts stored on the blockchain. Their central role makes it important to protect them from becoming overloaded, which could lead to them being unable to process valid requests. Any disruption to the API servers could cause significant impact to the supply chain process, leading to operational difficulties. To alleviate these risks, specific techniques can be employed to manage and control the traffic to the servers. One effective method is rate limiting, a process that limits the number of requests a user can make within a certain timeframe. This method helps prevent any user or entity from monopolizing the server's resources and degrading its performance. In addition, the use of DoS protection services can further enhance the server's resilience against attacks that try to overwhelm and incapacitate the system. These services are designed to identify and block traffic from sources exhibiting malicious or unusual behavior, thereby ensuring that the servers remain accessible to legitimate users.

The threat posed by authorized entities with potential malicious intent can be particularly challenging to address. Since these users are authorized, they have the capability to leak sensitive information or attempt to deliberately input false data into the system. Such actions can have serious consequences including privacy breaches and compromised data integrity. To reduce this risk, regular government-controlled audits should be performed to detect any misuse. Furthermore, strict penalties can be enforced for violations, reinforcing the consequences of system misuse. Additional strategies to address the potential issue of information leakage from the blockchain are elaborated in Section 6.2.

Physical attacks on the infrastructure hosting the API servers and nodes in the network pose a serious threat to the system's stability and security. These attacks can lead to service disruption, potential data loss, and breaches of sensitive information. The consequences of such incidents can extend beyond immediate operational difficulties, potentially damaging the trust in the system and causing reputational harm. To mitigate these risks, it is crucial to secure the physical locations housing the infrastructure. This can be achieved using access control mechanisms, surveillance systems, and security protocols. Moreover, the use of hard-drive encryption can

offer an additional layer of protection. Disk encryption will prevent attackers, who manage to gain physical access to a server or node, from accessing confidential and sensitive information.

In conclusion, the security analysis of SeaChain has identified several significant threats that need to be addressed before the system can be effectively deployed in a real-world scenario. These critical threats include unauthorized access, integrity of supply chain input data, vulnerabilities within smart contracts, the potential for DoS attacks, insider threats, and physical security risks. Each threat has been discussed in detail and appropriate mitigation strategies have been proposed. These strategies include enhancing access control, deploying reliable data registration systems, conducting thorough testing and audits of smart contracts, implementing rate limiting and DoS protection services, and enforcing physical security measures. Implementing these countermeasures will help ensure the confidentiality, integrity, and availability of SeaChain.

6.2 Comparison to related work

This section provides a comparative analysis of SeaChain and the related work reviewed in Section 2. The objective of this analysis is to contextualize the findings of this thesis, assess the unique contributions of the system, and identify potential areas for future development. The related work reviewed is primarily focused on the application of blockchain technology for enhancing supply chain management and product traceability. These studies provide valuable insights into state-of-the-art blockchain-based product traceability systems. Each work presents a unique approach to tackling the challenges of supply chain management, with each study emphasizing different aspects of importance. The comparison will focus on key aspects such as consumer accessibility, data privacy, scalability, and traceability.

One of the primary advantages of SeaChain is its enhanced accessibility for consumers. The system simplifies the process of viewing product information by allowing consumers to access data through a simple QR code scan. This feature significantly enhances the user friendliness and the amount of work required to view the data. In contrast, Wang *et al.* [3] require consumers to join the blockchain network as a node or lightweight node to view product data. This requirement could potentially limit the ease of access for consumers, as it is much more technical and cumbersome.

Ding *et al.* [10] and Salah *et al*. [14], on the other hand, do not present any specific method for consumers to view product data. Their focus primarily lies on the product traceability

framework itself, without mentioning how consumers interact with it. Madumidha *et al*. [11] propose the use of a dedicated app for displaying data to consumers. This approach is more consumer-focused than Wang *et al.* [3], but still involves more steps than scanning a QR code. Lin *et al.* [13] suggest that consumers can query product data using a product code and a smart contract address. Their explanation of this process is somewhat vague, leaving uncertainties around the ease of use for consumers. Finally, Malik *et al.* [12] propose using QR codes on products to retrieve product histories, similar to our approach. However, they do not provide a demonstration of the consumer interface, making it difficult to assess whether their system is as user-friendly as ours.

Our system, along with the ones proposed by Ding *et al.* [10] and Malik *et al.* [12], employs permissioned and private blockchain approaches. These blockchains restrict network access and safeguard sensitive enterprise data, contrasting with the public blockchain approaches used by the other systems reviewed in this thesis. The approach presented by Ding *et al.* [10] utilizes a combination of permissioned and private blockchains. This stands out as an optimal strategy for concealing sensitive data between enterprises and can provide valuable guidance for future improvements to our system.

To address scalability, several of the reviewed works utilize various strategies. Ding *et al.* [10] improve scalability by implementing a double-layer framework which separates the functions of data entry and data reading. This strategy allows for more efficient handling of data input and output requests. Malik *et al.* [12] employ sharding, which divides the blockchain into multiple separate chains, an approach that can increase the network's capacity. Lin *et al.* [13] uniquely address the challenge of rapid data accumulation in the blockchain. They manage this by dynamically balancing on-chain and off-chain data, which effectively reduces the volume of data stored directly on-chain. This mitigates the potential performance drawbacks that can arise due to large quantities of data in the blockchain. Each of these strategies offers a unique solution to help improve scalability. The data management scheme proposed by Lin *et al.* [13] could be a valuable addition if future large-scale testing indicates that data size might become an issue in our system.

Most of the reviewed product tracing systems, including SeaChain, utilize smart contracts in a similar way to track data throughout the supply chain. One exception is Malik *et al.* [12], which creates a transaction vocabulary that is used to track products. Based on the reviewed literature the use of smart contracts seems to be the best way to ensure accurate and efficient traceability

in blockchain-based systems. Therefore, the traceability functionality in SeaChain does not stand to gain any improvements from those presented in the reviewed literature.

The comparative analysis presented above provides a description of the unique attributes of SeaChain as well as its similarities with those described in the existing literature. Additionally, it emphasizes distinctive aspects of the other systems that could potentially improve our own. Our system surpasses the others in terms of consumer accessibility. Like the majority of the systems reviewed, SeaChain also utilizes smart contracts for product tracking. Data privacy could potentially be improved in our system by integrating elements of the double-layer framework proposed by Ding *et al.* [10]. Moreover, the scalability of our system, especially concerning data size, might be improved by adopting the dynamic data management approach proposed by Lin *et al.* [13]. It is important to note that like our system, all the others reviewed have only been tested as prototypes, with some systems being purely theoretical. To advance the development of blockchain-based product tracing it is essential to test these systems in real-world scenarios.

6.3 Reflection on thesis results

This section revisits the initial objectives of this research to assess its success. The first objective was to develop a proof-of-concept product tracing system using blockchain and smart contracts. The second objective was to evaluate if this system could improve the Norwegian fishing industry's supply chain and contribute to reducing fishery crimes. This section discusses these objectives considering our findings and evaluates if they were fulfilled. This assessment will help us determine the overall validity of our thesis statement.

Reading the design and implementation section should make it clear that the first objective has been achieved. A proof-of-concept product tracing system has been successfully implemented and evaluated. The system consists of a local GoQuorum blockchain network, three distinct smart contracts, and multiple API servers. Our findings indicate that, when compared to other systems in related literature, our system refines consumer accessibility and user-friendliness, while maintaining a high degree of traceability. Additionally, the research suggests that data privacy and scalability are potential areas for improvement in our system.

The evidence gathered from this research and the supporting literature indicates that the second objective has been achieved. SeaChain has the potential to significantly enhance the transparency, traceability, and security of data in the Norwegian fishing industry's supply chain.

By leveraging the system's collection of provenance data and the inherent immutability of blockchain, there is a strong potential to reduce fishery crimes. This could be achieved by ensuring authentic product origin and history, making illegal activities more difficult to perform without detection. However, to accurately measure the system's effectiveness in reducing fishery crimes, further research involving its deployment and use in real-world scenarios is required.

Given the successful achievement of both objectives, the findings of this research strongly support the thesis statement. The blockchain-based traceability system, leveraging smart contracts, has shown its potential to address the limitations of traditional supply chain management. The transparency and security provided by blockchain technology can enhance consumer confidence and reinforce regulatory compliance. Nonetheless, it is important to continue research in this field and perform real-world testing to verify and extend these findings.

7 Conclusion

7.1 Concluding remarks

We have successfully developed and evaluated SeaChain, a proof-of-concept product tracing system leveraging blockchain technology and smart contracts, specifically tailored for the Norwegian fishing industry. With SeaChain, we demonstrated the practical potential of blockchain in enhancing transparency, traceability, and security within supply chain management. Our research confirms our initial thesis and outlines a potential approach to alleviating the widespread issue of fishery crimes.

SeaChain provides enhanced consumer confidence by offering comprehensive product information, provenance data, and transaction histories for seafood products. This data is easily accessible to consumers through a simple QR code scan. The system also helps enforce regulatory compliance, making illegal activities more challenging to perform without detection. However, we must acknowledge the limitations of our system as well. Perhaps most importantly, the potential issues related to data privacy and scalability that we observed during the research. Furthermore, our security analysis of SeaChain identified certain threats that need to be addressed to ensure the robustness and reliability of the system.

Despite the limitations of SeaChain, this thesis provides a solid foundation for future research in this field, as well as a step towards strengthening and increasing traceability within the supply chain of the Norwegian fishing industry. Our findings extend beyond national context and the fishing industry, providing insights for global efforts to improve supply chain management.

7.2 Future work

Although the objectives of this thesis have been achieved, further research involving the deployment and use of the system in real-world scenarios is required to accurately measure its effectiveness in reducing fishery crimes. The issues related to data privacy need to be addressed and large-scale testing is necessary to reveal potential scalability issues. Furthermore, the mitigation strategies proposed in the security analysis should be implemented. Future work should aim to solve these issues and continue research in this field to extend and refine the findings of this thesis. The use of blockchain technology in supply chain management is a promising domain, and there is a lot of potential for future research.

References

[1] P. T. Aandahl and E. H. Brækkan. "Norge eksporterte sjømat for 151,4 milliarder kroner i 2022." https://seafood.no/aktuelt/nyheter/norge-eksporterte-sjomat-for-1514-milliarder-kroner-i-2022/ (accessed 2023).

[2] (2019). *Framtidens fiskerikontroll*. [Online] Available: https://www.regjeringen.no/no/dokumenter/nou-2019-21/id2680187/

[3] S. Wang, D. Li, Y. Zhang, and J. Chen, "Smart Contract-Based Product Traceability System in the Supply Chain Scenario," *IEEE Access*, vol. 7, pp. 115122-115133, 2019, doi: 10.1109/ACCESS.2019.2935873.

[4] A. Sharma *et al.*, "Up-to-the-Minute Privacy Policies via Gossips in Participatory Epidemiological Studies," (in English), *Frontiers in Big Data*, Original Research vol. 4, 2021-May-13 2021, doi: 10.3389/fdata.2021.624424.

[5] T.-A. S. Nordmo, A. B. Ovesen, H. D. Johansen, M. A. Riegler, P. Halvorsen, and D. Johansen, "Dutkat: A Multimedia System for Catching Illegal Catchers in a Privacy-Preserving Manner," presented at the Proceedings of the 2021 Workshop on Intelligent Cross-Data Analysis and Retrieval, Taipei, Taiwan, 2021. [Online]. Available: https://doi.org/10.1145/3463944.3469102.

[6] A. B. Ovesen, T.-A. S. Nordmo, H. D. Johansen, M. A. Riegler, P. Halvorsen, and D. Johansen, "File System Support for Privacy-Preserving Analysis and Forensics in Low-Bandwidth Edge Environments," *Information*, vol. 12, no. 10, p. 430, 2021. [Online]. Available: https://www.mdpi.com/2078-2489/12/10/430.

[7] T.-A. S. Nordmo *et al.*, "Njord: a fishing trawler dataset," presented at the Proceedings of the 13th ACM Multimedia Systems Conference, Athlone, Ireland, 2022. [Online]. Available: https://doi.org/10.1145/3524273.3532886.

[8] J. A. Alslie *et al.*, "Áika: A Distributed Edge System for AI Inference," *Big Data and Cognitive Computing*, vol. 6, no. 2, p. 68, 2022. [Online]. Available: https://www.mdpi.com/2504-2289/6/2/68.

[9] D. E. Comer *et al.*, "Computing as a discipline," *Commun. ACM*, vol. 32, no. 1, pp. 9–23, 1989, doi: 10.1145/63238.63239.

[10] Q. Ding, S. Gao, J. Zhu, and C. Yuan, "Permissioned Blockchain-Based Double-Layer Framework for Product Traceability System," *IEEE Access*, vol. 8, pp. 6209-6225, 2020, doi: 10.1109/ACCESS.2019.2962274.

[11] S. Madumidha, P. S. Ranjani, U. Vandhana, and B. Venmuhilan, "A Theoretical Implementation: Agriculture-Food Supply Chain Management using Blockchain Technology," in *2019 TEQIP III Sponsored International Conference on Microwave Integrated Circuits, Photonics and Wireless Networks (IMICPW)*, 22-24 May 2019 2019, pp. 174-178, doi: 10.1109/IMICPW.2019.8933270.

[12] S. Malik, S. S. Kanhere, and R. Jurdak, "ProductChain: Scalable Blockchain Framework to Support Provenance in Supply Chains," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, 1-3 Nov. 2018 2018, pp. 1-10, doi: 10.1109/NCA.2018.8548322.

[13] Q. Lin, H. Wang, X. Pei, and J. Wang, "Food Safety Traceability System Based on Blockchain and EPCIS," *IEEE Access*, vol. 7, pp. 20698-20707, 2019, doi: 10.1109/ACCESS.2019.2897792.

[14] K. Salah, N. Nizamuddin, R. Jayaraman, and M. Omar, "Blockchain-Based Soybean Traceability in Agricultural Supply Chain," *IEEE Access*, vol. 7, pp. 73295-73305, 2019, doi: 10.1109/ACCESS.2019.2918000.

[15] M. Asante, G. Epiphaniou, C. Maple, H. Al-Khateeb, M. Bottarelli, and K. Z. Ghafoor, "Distributed Ledger Technologies in Supply Chain Security Management: A Comprehensive Survey," *IEEE Transactions on Engineering Management*, vol. 70, no. 2, pp. 713-739, 2023, doi: 10.1109/TEM.2021.3053655.
[16] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 03/24 2009.
[17] E. Tedeschi, T.-A. S. Nordmo, D. Johansen, and H. D. Johansen, "On Optimizing Transaction Fees in Bitcoin using AI: Investigation on Miners Inclusion Pattern," *ACM Trans. Internet Technol.*, vol. 22, no. 3, p. Article 77, 2022, doi: 10.1145/3528669.
[18] B. H. Swathi, M. S. Meghana, and P. Lokamathe, "An Analysis on Blockchain Consensus Protocols for Fault Tolerance," in *2021 2nd International Conference for Emerging Technology (INCET)*, 21-23 May 2021 2021, pp. 1-4, doi: 10.1109/INCET51464.2021.9456310.
[19] S. J. Alsunaidi and F. A. Alhaidari, "A Survey of Consensus Algorithms for Blockchain Technology," in *2019 International Conference on Computer and Information Sciences (ICCIS)*, 3-4 April 2019 2019, pp. 1-6, doi: 10.1109/ICCISci.2019.8716424.
[20] V. Buterin. "Ethereum Whitepaper." https://ethereum.org/en/whitepaper/ (accessed 02.04, 2023).
[21] E. Tedeschi, T. A. S. Nordmo, D. Johansen, and H. D. Johansen, "Predicting Transaction Latency with Deep Learning in Proof-of-Work Blockchains," in *2019 IEEE International Conference on Big Data (Big Data)*, 9-12 Dec. 2019 2019, pp. 4223-4231, doi: 10.1109/BigData47090.2019.9006228.
[22] "Ethereum development documentation." https://ethereum.org/en/developers/docs/ (accessed 2023).
[23] E. Lin. "Consensus protocols." https://docs.goquorum.consensys.net/concepts/consensus (accessed 02.04, 2023).
[24] "Solidity." https://docs.soliditylang.org/en/v0.8.19/ (accessed 2023).
[25] "web3.js - Ethereum JavaScript API." https://web3js.readthedocs.io/en/v1.8.2/ (accessed 2023).
[26] "go-ethereum - Official Go implementation of the Ethereum protocol." https://geth.ethereum.org/ (accessed 2023).
[27] H. Liu, X. Luo, H. Liu, and X. Xia, "Merkle Tree: A Fundamental Component of Blockchains," in *2021 International Conference on Electronic Information Engineering and Computer Science (EIECS)*, 23-26 Sept. 2021 2021, pp. 556-561, doi: 10.1109/EIECS53707.2021.9588047.
[28] V. Alvarez, S. Richter, X. Chen, and J. Dittrich, "A comparison of adaptive radix trees and hash tables," in *2015 IEEE 31st International Conference on Data Engineering*, 13-17 April 2015 2015, pp. 1227-1238, doi: 10.1109/ICDE.2015.7113370.
[29] S. G. a. J. Dean. "LevelDB." https://github.com/google/leveldb (accessed 2023).
[30] E. Lin. "Architecture." https://docs.goquorum.consensys.net/concepts/architecture (accessed 2023).
[31] N. Oba. "Finality in Blockchain." https://medium.com/minima-global/finality-in-blockchain-e5a62ca0f9f4 (accessed 2023).
[32] E. Lin. "Comparing proof of authority consensus protocols." https://docs.goquorum.consensys.net/concepts/consensus/comparing-poa/ (accessed 2023).
[33] J. R. Douceur, "The Sybil Attack," in *Peer-to-Peer Systems*, Berlin, Heidelberg, P. Druschel, F. Kaashoek, and A. Rowstron, Eds., 2002// 2002: Springer Berlin Heidelberg, pp. 251-260.

[34] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," presented at the Proceedings of the third symposium on Operating systems design and implementation, New Orleans, Louisiana, USA, 1999.
[35] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: BFT Consensus with Linearity and Responsiveness," presented at the Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, Toronto ON, Canada, 2019. [Online]. Available: https://doi.org/10.1145/3293611.3331591.
[36] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, 1982, doi: 10.1145/357172.357176.
[37] ConsenSys. "Scaling Consensus for Enterprise: Explaining the IBFT Algorithm." https://consensys.net/blog/enterprise-blockchain/scaling-consensus-for-enterprise-explaining-the-ibft-algorithm/ (accessed 2023).
[38] E. Lin. "Configure IBFT consensus." https://docs.goquorum.consensys.net/configure-and-manage/configure/consensus-protocols/ibft (accessed 2023).
[39] IBM. "What is Docker?" https://www.ibm.com/topics/docker (accessed 2023).
[40] "Docker Compose overview." https://docs.docker.com/compose/ (accessed 2023).
[41] R. Hu, W. Ding, and L. Yang, "A survey on data provenance in IoT," *World Wide Web*, vol. 23, 03/01 2020, doi: 10.1007/s11280-019-00746-1.
[42] FAO. "ASFIS List of Species for Fishery Statistics Purposes." https://www.fao.org/fishery/en/collection/asfis/en (accessed.
[43] (2023). *GS1 General Specifications Standard*. [Online] Available: https://ref.gs1.org/standards/genspecs/