INFORMATICS INSTITUTE OF TECHNOLOGY

INFORMATICS INSTITUTE OF TECHNOLOGY

In Collaboration with

UNIVERSITY OF WESTMINSTER

Proof of Identity—a Blockchain Consensus Algorithm to Improve Byzantine Fault Tolerance in Swarm Robotics

An interim progress report by

Mr. Theviyanthan Krishnamohan

Supervised by

Mr. Ragu Sivaraman

Submitted in partial fulfillment of the requirements for the MSc in Advanced Software Engineering degree at the University of Westminster.

August 2022

Declaration

I, Theviyanthan Krishnamohan, hereby certify that this dissertation and all the artifacts produced by this research work are my own work unless specified otherwise and wherever others' work has been used, it is properly cited and the authors are given due credit. This research work has not been/is not being submitted to any other institutions.

Name of the Student: Theviyanthan Krishnamohan

Registration Number: 20201022/w1839957

Signature:  Date: 1st August 2022

Abstract

Swarm robotics applies concepts of swarm intelligence to robotics. Discrete consensus achievement is one of the major behaviors found in swarm robotics. Various algorithms have been developed for discrete consensus achievement. However, existing discrete consensus-achievement algorithms, referred to henceforth as classical solutions, are vulnerable to Byzantine robots. Blockchain has been successfully used to mitigate the negative effect of Byzantine robots in discrete consensus achievement. Blockchain is the technology behind cryptocurrencies that allows the creation of immutable, decentralized, and distributed ledgers. Nevertheless, since the blockchain solution uses the Proof-of-Work blockchain consensus algorithm, it is vulnerable to the 51% attack. Besides, the swarm also takes longer to achieve consensus.

This research proposes a novel blockchain consensus algorithm called Proof-of-Identity—which uses a private-public key pair and a swarm controller—to create a dynamically permissioned blockchain that would negate the 51%-attack problem associated with the Proof-of-Work algorithm while also reducing the consensus time.

This proposed solution was tested against the classical solution and the existing blockchain solution using the collective perception scenario. The collective perception scenario is used to benchmark different discrete consensus achievement algorithms. Test results show that the Proof-of-Identity algorithm prevents the 51%-attack problem while improving the consensus time in comparison to the existing blockchain solution without affecting the exit probability.

Keywords—Swarm robotics, blockchain, Proof of Work, consensus algorithm, consensus achievement, collective perception scenario.

Publications

1. **A Review of the Consensus Achievement Strategies in the Context of Swarm Robotics**

   A review paper that studies the consensus achievement strategies against the backdrop of swarm robotics and critically analyzes their limitations.

   Journal: Swarm and Evolutionary Computation, Elsevier

   Status: Under review

2. **Improving Byzantine Fault Tolerance in Swarm Robotics Collective Decision-Making Scenario via a New Blockchain Consensus Algorithm**

   A research paper that dwells on the collective perception experiment carried out using the proposed Proof of Identity algorithm.

   Journal: Swarm Intelligence, Springer Nature

   Status: Under review

3. **Proof of Identity—a Blockchain Consensus Algorithm to Create a Dynamically Permissioned Blockchain**

   A research paper that elucidates the Proof-of-Identity algorithm and presents the results of the tests carried out against the Proof-of-Work algorithm.

   Journal: International Journal of Blockchains and Cryptocurrencies

   Status: Under review

4. **A Survey of Applications of Blockchain in Collective Decision-Making Scenarios in Swarm Robotics**

   A review paper that critically analyzes the application of blockchain in the collective decision-making scenarios in swarm robotics.

   Journal: Ledger

   Status: Under review

5. **Analyzing the Suitability of Blockchain Consensus Algorithms in Swarm Robotics**

   A review paper that studies the existing blockchain algorithms and their suitability in swarm robotics.

   Journal: IET Blockchain

   Status: Under review

Acknowledgment

The author would like to extend his gratitude to the supervisor of this research Mr. Ragu Sivaraman for his invaluable guidance and mentorship. The author is also heavily indebted to the project coordinator Mr. Guhanathan Poravi for reviewing the progress of the thesis and providing steadfast support and moral guidance during the course of this project. The author would also like to express his gratitude to Dr. Volker Strobel for assisting in gathering requirements and for evaluating this research work. Finally, the author would like to thank their employer for partially funding this scholarship and for being flexible with the work and the working hours to facilitate this research work.

Table of Contents

List of Figures

List of Tables

List of Equations

Abbreviations

Abbreviation	Full form
PoW	Proof of Work
PoI	Proof of Identity
DMMD	Direct Modulation of Majority-based Decision
DMVD	Direct Modulation of Voter-based Decision
DC	Direct Comparison
LCP	Linear Consensus Protocol
W-MSR	Weighted Mean Subsequence Reduced
PoA	Proof of Authority
DAO	Decentralized Autonomous Organization
CLI	Command-Line Interface
GUI	Graphical User Interface
CSV	Comma-Separated Values

Chapter 1 Introduction

1.1 Chapter Overview

This chapter discusses the problem domain in detail, defines the problem, studies existing research works, and identifies research gaps. Subsequently, this document lays down the objectives of this research project, its deliverables, and scopes.

1.2 Problem Domain

1.2.1 Swarm Robotics

Swarm robotics is an approach whereby the principle of swarm intelligence is applied to robotics to solve problems that cannot be solved by single, monolithic robots or multi-agent robots.

Swarm intelligence draws inspiration from biological systems in nature such as bee colonies, ant colonies, bacterial growth, and bird flocking. Such systems involve the coordination of simple individuals to solve complex problems. For instance, insect societies consist of simple, nearly homogenous units that are decentralized and not synchronized and communicate with one another using pheromones to find the best path to a source using a positive feedback mechanism (Beni, 2005).

Robots can be defined as entities “capable of both mechanical and informational behavior”. Swarm robotics employ simple robots to mimic simple individuals found in biological swarms and can be used to solve real-life problems—that are difficult to solve by other means—using swarm intelligence.

Swarm robotics is formally defined as “the study of how large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among agents and between the agents and the environment” (Şahin, 2005).

The many advantages of swarm robotics such as being able to mass-produce robots owing to their simplicity, and the reliability that stems from their redundancy mean that this technology

can be used for tasks such as those “that cover a region”, “that are dangerous”, “that scale up or down in time”, and “that require redundancy (Beni, 2005)(Şahin, 2005).

1.2.2 Classification of Swarm Engineering

Swarm engineering is defined as “a fusion of dependable systems engineering and swarm intelligence” (Winfield, Harper and Nembrini, 2005). A review carried out by Brambilla et al. (2013) classifies the existing works into two major taxonomies, namely methods and collective behaviors (Brambilla *et al.*, 2013).

The methods taxonomy classifies published works based on the methods used to design swarm robotics systems and, thus, is of less interest to this study. The collective-behaviors taxonomy reviews the basic behaviors exhibited by swarms to address real-world challenges.

Collective behaviors are categorized into four main groups: spatially organizing behaviors, navigation behaviors, collective decision-making behaviors, and other collective behaviors.

The behaviors of interest to this research study are the collective decision-making behaviors. Collective decision-making is the process of having a swarm of robots collectively agree on a single decision.

According to Brambilla et al. (2013), this behavior can be exploited to address two different requirements, viz. consensus achievement, and task allocation. Consensus achievement is the behavior of reaching an agreement on one choice among several other alternatives whereas task allocation is the behavior of robots distributing different tasks among themselves.

Of these two behaviors, this research focuses on the consensus-achievement behavior. Consensus achievement is imperative in applications of swarm robotics such as identifying radiation leakage and oil spillage and is inspired by the behaviors of insect species such as ants that collectively identify the shortest path with the use of pheromones, and bees that collectively decide the best nesting or foraging location (Camazine, Deneubourg, Franks, 2001)(Couzin *et al.*, 2005).

This research study attempts to build on the collective perception scenario proposed by Valentini et al. to compare the generalizability of three consensus-achievement strategies,

namely the Direct Modulation of Majority-based Decisions (DMMD), the Direct Modulation of Voter-based Decisions (DMVD), and the Direct Comparison (DC) (Valentini, Brambilla, *et al.*, 2016).

1.3 Problem Definition

The critical analysis of the existing research works that attempt to solve the collective perception scenario, referred to henceforth as classical solutions, identified two major issues—the absence of Byzantine fault tolerance and an authentication mechanism to allow robots to authenticate themselves. Besides, works that successfully address the aforementioned issues to a limited extent do so at the expense of performance.

The literature survey also found blockchain to offer a promising space to address concerns with regard to Byzantine fault tolerance. However, solutions that utilize blockchain to achieve consensus port algorithms used in non-blockchain solutions to blockchain, failing to maximize the benefits afforded by blockchain and, thus, compromising on the performance of the solution. Furthermore, the use of the de-facto Proof-of-Work (PoW) consensus algorithm severely impacts the time taken to achieve consensus in the swarm while also making the swarm susceptible to the 51% attack (Saad *et al.*, 2020).

A survey on the security challenges in swarm robotics by Higgins, Tomlinson and Martin (2009) found identity and authentication, and intrusion detection to be major security challenges (Higgins, Tomlinson and Martin, 2009). Such challenges could pose a major threat, especially in critical applications of consensus achievement such as detecting radiation leakage and the use of swarms of micro aerial vehicles in the military since bad actors can introduce Byzantine robots to compromise the functionality of the system and cause grave harm to human lives.

Thus, it is paramount that the aforementioned security issues are addressed without compromising on performance in order to be able to use swarm robotics to solve real-life problems.

1.3.1 Problem Statement

Classical solutions to consensus achievement in swarm robotics are susceptible to Byzantine robots while blockchain-based solutions do not offer complete Byzantine fault tolerance—due to their vulnerability to the 51% attack—and suffer from comparatively poor performance.

1.4 Research Motivation

The author is interested in both blockchain, and robotics, and this research work straddles both of these domains. The author has already published a research paper on using blockchain to decentralize the control plane in Software Defined Networks (SDN) and this research will enable the author to further explore the realm of blockchain.

1.5 Related Work

1.5.1 Existing Applications of Blockchain in Consensus Achievement in Swarm Robotics

The existing works have been summarized in Table 1. The existing works mostly use the Proof-of-Work algorithm which has resulted in the solutions performing worse in comparison to the classical consensus strategies. The only work that proved to have a better performance than the classical strategies was vulnerable to Byzantine robots.

Existing Work	Consensus Algorithm Used	Is Vulnerable to Byzantine Robots?	Performs Better Than Classical Strategies?	Is the Solution Generalizable?	Limitations
Strobel, Castelló Ferrer and Dorigo (2020)	PoW	No	No	Yes	This addresses continuous consensus achievement, but the solution is not applicable to discrete consensus achievement.

Singh et al. (2020)	Proof of Authority (PoA)	Yes	N/A	No	Violates swarm robotics principles since the validators centralize the functionality of the system. Moreover, the solution does not address the Byzantine problem.
Nguyen, Hatua and Sung (2020)	PoW	Yes	Yes	No	The solution is vulnerable to Byzantine robots and the generalizability of the solution was not shown.
Strobel, Ferrer and Dorigo (2018)	PoW	No	No	Yes	Ports the classical strategies directly to blockchain. Vulnerable to the 51% attack and performs poorly in comparison.

Table 1 A summary of existing applications of blockchain in consensus achievement in swarm robotics

1.6 Research Gap

A critical analysis of the existing literature on the blockchain-based consensus achievement strategies revealed several gaps, out of which this research work attempts to address the following gaps:

- Blockchain-based solutions perform poorly in comparison to the classical solutions due to the use of the PoW algorithm.
- Blockchain-based solutions do not offer complete Byzantine fault tolerance since the PoW algorithm makes the solution vulnerable to the 51% attack.

1.7 Research Contribution

1.7.1 Contribution to Technology

This research contributes a new consensus algorithm to the blockchain realm. Since one of the goals of the new consensus algorithm is to allow legitimate robots to take part in mining, this consensus algorithm can be used in blockchain networks where only a selected set of miners should be allowed to mine new blocks. This consensus algorithm differs from the Proof-of-Authority algorithms by allowing the dynamic addition of miners without a voting process. In the Proof-of-Authority algorithms, either a list of miners must be specified during the genesis, or the miners must be voted in by other miners while no specifications exist to decide the basis of a vote. This makes it difficult to add new legitimate miners during the runtime. In contrast,

the new consensus algorithm allows new legitimate miners to be added to the network during the runtime easily.

1.7.2 Contribution to Domain

This research allows swarm robotics to be more practical for real-life use cases by addressing the Byzantine-robot problem using blockchain and providing protection against the 51% attack by way of a new consensus algorithm. Strobel, Ferrer and Dorigo (2018) have already demonstrated the advantages of using blockchain in swarm robotics (Strobel, Ferrer and Dorigo, 2018). By improving the performance of the swarm when blockchain is used to be on par with the PoW-based solutions, this research work also nullifies the performance advantage PoW-based solutions may have over other blockchain solutions.

1.8 Research Challenge

Both the vulnerability to the 51% attack and the poor performance in comparison to the classical solutions are due to the use of the PoW consensus algorithm in blockchain. In order to address this issue, a different consensus algorithm should be used. However, other existing consensus algorithms such as Proof of Authority, Proof of Stake, Proof of Cooperation, etc. are not suitable to be used in swarm robotics.

Thus, the research challenge is to develop a consensus algorithm for swarm robotics that will

1. Make the system more Byzantine fault-tolerant by not being vulnerable to the 51% attack.
2. Ensure the improved security does not come at the cost of performance.
3. Be consistent with the principles of swarm robotics by allowing any legitimate robot to join the swarm as a miner at any time.

The Ethereum blockchain platform that will be used in this research project is open-source and, thus, a new consensus algorithm can be developed on top of the Ethereum blockchain platform. This makes achieving the objectives of this research feasible.

1.9 Research Questions

1. How can the security of blockchain-based consensus achievement strategies be improved while not compromising on performance?
2. How can the blockchain-based consensus strategies be made more Byzantine fault-tolerant?
3. How can new legitimate robots be allowed to join the swarm as miners while making the swarm immune to Byzantine robots?

1.10 Research Aim

The aim of this research is to design, develop, and evaluate a new blockchain consensus algorithm that will make the swarm more tolerant of Byzantine robots by resolving the 51%-attack vulnerability associated with the PoW algorithm, and allow legitimate robots to authenticate themselves with the swarm as miners without affecting the exit probability and consensus time associated with the PoW algorithm.

1.11 Research Objectives

Objective	**Description**	**Learning Outcomes**
Literature Survey	• Critically reviewing the new developments in the field of swarm robotics. • Studying the various branches of swarm robotics. • Critically analyzing existing literature to identify research gaps. • Studying the application of blockchain in swarm robotics. • Critically analyzing existing literature on the use of blockchain in swarm robotics. • Reviewing the technologies used in swarm-robotics-related research works. • Evaluating the technologies that are used in swarm robotics.	LO1

Requirement Gathering	• Identifying the development platforms needed to build a new blockchain consensus algorithm. • Identifying the most appropriate blockchain framework • Identifying the most appropriate simulation environment for swarm robotics • Determining the functionality of the new blockchain consensus algorithm • Determining the algorithms that might be needed to develop the new consensus algorithm	LO5, LO3
Designing	• Designing a new blockchain consensus algorithm • Designing appropriate evaluation strategies to compare the performance against classical strategies • Designing an interface between blockchain and the swarm robotics simulation environment.	LO3, LO4
Developing	• Developing a new blockchain consensus algorithm • Developing appropriate evaluation strategies to compare the performance against classical strategies • Developing an interface between blockchain and the swarm robotics simulation environment.	LO3, LO2, LO5
Evaluating	• Evaluating how much the swarm can tolerate Byzantine robots using the new consensus algorithm. • Evaluating how well the swarm performs in comparison to classical strategies and blockchain-based strategies in terms of exit probability and consensus time.	LO3, LO5
Documenting the work	• Producing a thesis documenting the research work and its findings. • Producing a research paper summarizing the proposed solution, evaluation, and analysis.	LO5, LO2, LO6, LO4

Table 2 Research objectives

1.12 Project Scopes

1.12.1 In Scope

- Developing a new blockchain consensus algorithm to address the 51%-attack problem without compromising on performance.
- Evaluating the new consensus algorithm on a generalized discrete consensus achievement problem.
- Allowing legitimate robots to join the swarm as miners.

1.12.2 Out of Scope

- Protecting the swarm against Sybil attacks
- Evaluating the consensus algorithm on continuous consensus achievement problems.
- Developing blockchain-native consensus strategies to improve performance.
- Allowing legitimate robots to authenticate themselves with the swarm to participate in consensus achievement.

1.12.3 Prototype Diagram

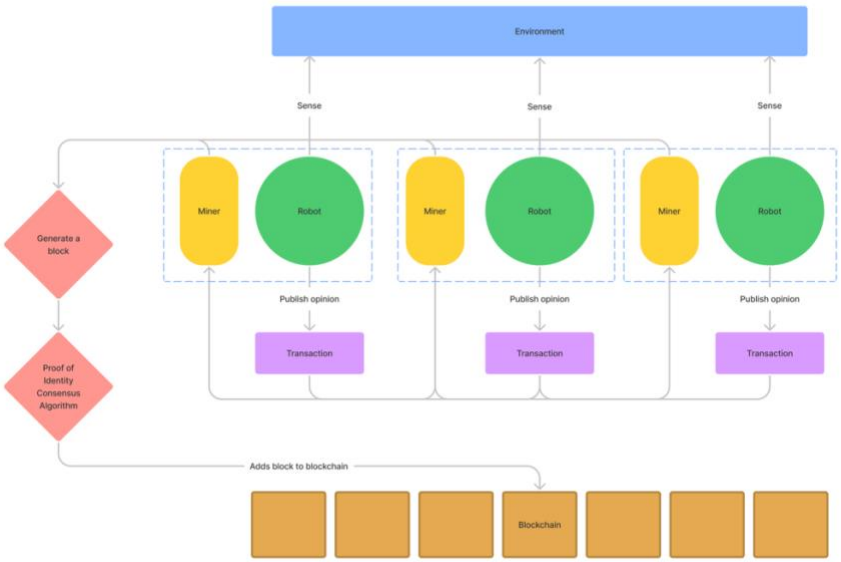


Figure 1 A diagram showing the high-level functionality of the prototype.

1.13 Chapter Summary

This chapter examined the problem domain and defined the problem before analyzing the existing research works. Further, this chapter also identified the research gaps. Consequently, this document discussed the objectives of this research project, its deliverables, and scopes.

Chapter 2 Literature Review

2.1 Chapter Overview

Swarm robotics is a field that draws inspiration from biological systems by attempting to solve real-life problems collectively using a swarm of simple robots. Collective decision-making is one of the basic behaviors of swarm robotics and can be used for both task allocation and consensus achievement. Previous research works have introduced several algorithms for achieving consensus in swarm robotics. However, these algorithms do not take Byzantine robots into account and, thus, are not Byzantine fault-tolerant. In recent times, researchers have explored the possibility of using blockchain to achieve consensus in swarm robotics and to address the problem of Byzantine robots.

Nevertheless, the use of the de-facto Proof-of-Work (PoW) algorithm in blockchains has made the blockchain-based solutions less performant in comparison to the classical, non-blockchain solutions. Besides, existing blockchain-based solutions do not attempt to resolve the Byzantine problem introduced by means of the 51% attack.

This chapter reviews the domain of swarm robotics, studies the collective perception scenario, critically analyzes the existing solutions to the collective perception scenario, reviews blockchain and the application of blockchain in swarm robotics, analyzes blockchain consensus algorithms, and studies the evaluation and benchmarking frameworks.

2.2 Concept Map

The concept map of the literature review can be found in the Appendix section under **Concept Map**.

2.3 Problem Domain

2.3.1 Introduction to Swarm Robotics

Swarm robotics tries to solve problems, which cannot be solved by single, monolithic robots or multi-agent robots, by using the principle of swarm intelligence. Swarm intelligence is a discipline that takes inspiration from biological systems found in nature such as bee colonies, bacterial growth, ant colonies, and bird flocking.

In swarm intelligence, complex problems are solved through coordination among simple individuals. For example, simple and almost homogenous individual insects in insect societies, which are decentralized and not synchronized, have been shown to be communicating with one another using pheromones to find the best path to a source using a positive feedback mechanism (Beni, 2005).

Robots are devices "capable of both mechanical and informational behavior". Swarm robotics makes use of multiple simple robots to imitate simple individuals found in swarms in nature and can be utilized to solve real-life problems that cannot be easily solved by other means.

Consequently, swarm robotics is formally defined as "the study of how large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among agents and between the agents and the environment" (Şahin, 2005).

Even though there does not exist a consensus in the literature on the characteristic features of swarm robotics, the following characters are commonly found in most of the literature (Zakiev et al., 2018).

- A large number of members
- Simple members
- Homogeneity
- Decentralization
- Distribution
- Local sensing and communication

The many advantages of swarm robotics such as being able to mass-produce robots owing to their simplicity and the reliability that stems from their redundancy mean that this technology can be used for tasks such as those "that cover a region", "that are dangerous", "that scale up or down in time", and "that require redundancy(Beni, 2005)(Şahin, 2005).

Environmental monitoring for oil leakage or nuclear radiation is an example of tasks that cover a region. Tasks that are dangerous are activities like demining minefields and firefighting. Swarm robotics is suitable for such tasks because individual robots cost less by virtue of their

simplicity. Besides, the loss of a few robots will not affect the swarm, thereby ensuring reliability (Şahin, 2005).

Oil leakage from a sunken ship is an excellent example of tasks that scale up or down in time. When swarm robotics is used to contain oil spillage, more robots can be deployed to the swarm if the oil leakage increases. An example of tasks that require redundancy is the use of swarm robotics on battlefields. A swarm of robots will continue to function even if a few robots are hit by bullets and are destroyed (Şahin, 2005).

2.3.2 Classification of Swarm Engineering

Winfield et al. (2005) define swarm engineering as "a fusion of dependable systems engineering and swarm intelligence". Brambilla et al. (2013) classified swarm engineering based on existing works into two main taxonomies, namely methods and collective behaviors.

The methods used to design swarm robotics systems form the basis of the methods taxonomy and, thus, are of little importance to this study. Behaviors of swarms to solve real-world problems are used to form the collective-behaviors taxonomy.

Collective behaviors, in turn, are divided into four major groups, namely spatially organizing behaviors, navigation behaviors, collective decision-making behaviors, and other collective behaviors.

This study focuses on the collective decision-making behaviors. Brambilla et al. (2013) defined collective decision-making as "how robots influence each other when making choices". Thus, collective decision-making is a process by which multiple individuals in a swarm come to an agreement on a decision.

Brambilla et al. (2013) argued that collective decision-making behavior can be used to perform two different functions, namely consensus achievement, and task allocation. Consensus achievement is the behavior of reaching an agreement on one decision among many others while task allocation is the behavior of robots delegating different tasks among themselves.

Of these two functions, this research concentrates on the consensus-achievement function. Consensus achievement is paramount in applications of swarm robotics such as detecting radiation leakage and oil spillage and draws inspiration from the behaviors of bees that collectively decide the best nesting or foraging location and insect species such as ants that collectively identify the shortest path with the use of pheromones. (Camazine, Deneubourg, Franks, 2001)(Couzin *et al.*, 2005).

2.3.3 Consensus Achievement

Consensus achievement is having a swarm of robots come to an agreement on one choice among several other choices.

Valentini et al. (2017) further classified the consensus achievement behavior into discrete and continuous consensus achievement. This classification is based upon the "cardinality of the choices available to the swarm". The consensus achievement problem is discrete if the choices available to a swarm are finite and countable. On the other hand, if the choices available are infinite and measurable, then the consensus achievement problem is said to be continuous.

In order to create an abstraction of "the structure and logic of discrete consensus achievement", (Valentini et al., 2017) introduced the best-of-n problem in their paper. Simply put, the best-of-n problem is choosing an option i among n number of options so that the quality of i is maximized and the cost of i is minimized. The quality and cost associated with an option are both functions of their environment.

Each robot in the swarm picks an option *i,* where $i \in \{1, 2, \ldots n\}$. A collective decision is said to have been made when a large majority of the robots, characterized by m, where $m \geq (1 - x)n$ and $0 \leq x \leq 0.5$, agree on an opinion. Here, x is a threshold value set by the designer of the problem. If $x = 0$, then it becomes a consensus decision.

The authors further introduced taxonomies predicated upon the symmetricity of the quality and cost of the best-of-n problem. If all the options available are of equal quality, the quality is symmetric, which also holds true for cost. If there are options of different quality (or cost), then the quality (or cost) is said to be asymmetric.

Based on the symmetric and asymmetric nature of the cost and quality, Valentini et al. (2017) classified the best-of-n problem into symmetric option qualities and costs, symmetric option qualities and asymmetric option costs, asymmetric option qualities and symmetric option costs, asymmetric option qualities and costs: synergic case, and asymmetric option qualities and costs: antagonistic case.

2.3.4 Collective Perception

Valentini, Brambilla, et al. (2016), in a subsequent paper, argued that the effectiveness of a collective decision-making strategy can not only be decided by the accuracy of the decision and the time taken to make that decision but it can also be decided by the generality of the strategy. A general strategy will allow engineers to reuse the high-level control logic of the strategy to address varying problems.

The authors, in their seminal paper, consequently, introduced a novel decision-making scenario termed as the collective perception problem to test the generality of different decision-making strategies (Valentini, Brambilla, *et al.*, 2016).

The collective perception problem is a discrete best-of-n problem where a swarm of robots has to come to an agreement on which of the two colors of black and white is most frequently found in the environment. To draw a real-life parallel, the collective perception problem is tantamount to a swarm of robots evaluating “the availability of precious metals”, or “the presence of pollutants or cancer cells” in an environment.

In the collective perception scenario, there exists a square area of area $200 \times 200\text{cm}^2$ consisting of a grid that contains cells each of area $10 \times 10\text{cm}^2$. The square area is bounded by walls and can be detected by the robots to avoid collisions.

Each cell in the square area is either colored black or white, with each color being an abstraction of a feature in an environment. Thus, the collective perception scenario is a best-of-2 problem since the swarm needs to choose one of the two available options.

The quality of the color is determined by its frequency. In the collective perception problem, the authors set black to be the most frequent color. Hence, the swarm is expected to choose black as the best option.

E-puck robots are used as the swarm members in this problem. The e-puck robots are equipped with sensors to both avoid collisions and to detect the brightness of the floor, which they can then use to determine the color (Mondada *et al.*, 2009).

The smaller footprint (a diameter of 7cm) of the robots ensures that the environment is orders of magnitude larger than a single robot. Besides, an e-puck comes equipped with RGB LEDs, an accelerometer, a sound sensor, a low-resolution camera, and 8 proximity sensors. The robot also has three ground sensors that are used to measure the gray-scale values of the surface. Its battery can last up to 45 minutes and the robot has the ability to move at $16cms^{-1}$.

In the collective perception scenario, the robots start with an initial opinion. The red LED is lit if their opinion is black, and the blue LED is lit if their opinion is white.

Since its publishing, this collective perception scenario has been used as a testing ground by subsequent researchers to test different consensus achievement strategies.

2.3.5 Consensus Achievement Strategies

Valentini, Brambilla, et al. (2016) used the collective perception scenario to test three different decision-making strategies that they had developed. The three strategies employed a set of common routines to find the best color and differed only in the way they chose the best option. The common routines are discussed in detail below.

There are two states in these strategies, namely the exploration state and the dissemination state and these are akin to the waggle dance found in the bee populations (Frisch, 1993).

In the exploration state, as the name implies, the robots explore their environments by performing random walk and rotations. A robot moves in straight lines for a random time sampled from an exponential distribution with a mean value of 40s. Then, the robot turns for a random time chosen from a uniform distribution, which is between 0s to 4.5s. The direction of

rotations is randomly chosen with both clockwise and counterclockwise rotations having an equal probability. Once the rotation is complete, the robots continue their linear motion.

If a robot stumbles upon an obstacle, either in the way of another robot or a wall, within approximately 30cm, the robot pauses its motion, and the obstacle avoidance routine is triggered.

During the obstacle avoidance routine, a robot utilizes its proximity sensors to detect the distance and bearing of each identified obstacle. It, then, makes use of this information to compute a new direction opposite to the obstacles. Following this, the robot starts its random walk again.

During this exploration state, the robots use their ground sensors to measure the quality of their opinion. To measure the quality of their opinion, the robots sample the color of the surface as they perform random walk. The quality p_i of an opinion i, where $i \in \{a, b\}$ (a corresponds to black and b to white), is defined as the fraction of the amount of time the robot perceived the color of its opinion (t_i) over the total amount of time the robot spent in the exploration state (t).

$$p_i = \frac{t_i}{t}$$

Equation 2.1

For instance, if a robot's opinion is black (opinion a) and it spends 20s in the exploration state, during which it perceives the color black on the floor for 10s, then the quality of its opinion p_a is given by:

$$p_a = \frac{10s}{20s} = 0.5$$

Equation 2.2

This perceived quality is the metric based on which the swarm decides the best opinion.

The exploration state is followed by the dissemination state. During the dissemination state, a robot engages in random walk, rotation, and obstacle avoidance as it does during the exploration state.

However, in addition to these general routines, the robot also broadcasts its opinion to its neighbors. As to what else happens during this state is determined by the decision-making strategy used.

2.3.5.1 Direct Modulation of Majority-based Decision (DMMD)

When the DMMD strategy is used, a robot remains in the dissemination state for a random amount of time sampled from an exponential distribution with a mean given by the quality of the robot's opinion p_i times a constant g. Here, g is a parameter whose value is decided by the designer. Thus, the higher the quality of opinion p_i, the longer the robot remains in the dissemination state. This ensures that a robot with a higher quality opinion gets to broadcast its opinion to a lot of neighbors.

In this strategy, p_i acts as a modulator, thus, giving this strategy a part of its name, and g acts as an unbiased dissemination time.

During the dissemination state, robots broadcast their opinion p_i to their neighbors while receiving the opinions of the neighboring robots. Towards the end of the dissemination time, a robot changes its opinion to the opinion of the majority, which includes that of its own, and then switches back to the exploration state (Valentini, Hamann and Dorigo, 2015)(Valentini, Ferrante, *et al.*, 2016).

2.3.5.2 Direct Modulation of Voter-based Decision (DMVD)

The DMVD strategy is similar to DMMD and differs only in its decision-making mechanism. Much like the DMMD strategy, the DMVD strategy also modulates its dissemination time based on the perceived quality of its opinion p_i.

Nonetheless, when the DMVD strategy is used, robots adopt the opinion of a random neighbor as their own (Valentini et al., 2014).

2.3.5.3 Direct Comparison (DC)

The DC strategy does not modulate the dissemination time like the DMMD and DMVD strategies. Instead, the dissemination time is a random sample drawn from an exponential distribution with a mean of g.

Moreover, the robots not only broadcast their own opinion to their neighbors, but they also broadcast the quality of their own opinion. At the end, robots compare the quality of their opinion with that of a random neighbor and adopt the greater of the two as their own opinion (Valentini, Brambilla, *et al.*, 2016).

Consensus is said to have been achieved when all the robots end up having the same opinion.

2.3.6 Evaluation Metrics

Valentini, Brambilla, et al. (2016) compared the three strategies discussed above using the collective perception scenario. In order to analyze the performance of the decision-making strategies, the authors used the exit probability E_N and consensus time T_N of the strategies.

The exit probability is the number of times a strategy resulted in the correct consensus over the total number of times the experiment was run. The consensus time is the average amount of time taken for a correct consensus.

2.3.7 Experiment Parameters

The authors adjusted the difficulty of the scenario by adjusting the number of black and white cells. The difficulty p_b^* of choosing black as the best opinion is given by:

$$p_b^* = \frac{p_b}{p_a}$$

Equation 2.3

Where:

$$p_a = \frac{n_a}{n_a + n_b}$$

Equation 2.4

$$p_b = \frac{n_b}{n_a + n_b}$$

Equation 2.5

Where n_a is the number of black cells and n_b is the number of white cells on the floor. In other words, the difficulty is the ratio between the percentage of black cells and white cells on the floor.

Furthermore, the initial number of robots with opinion *a* ($E_a(0)$) was also adjusted to study their impact on the performance, and the size of the swarm was set to 20 robots.

The authors used the ARGoS simulator to carry out the experiment (Pinciroli *et al.*, 2012). Two difficulty values of 0.515 and 0.923 were used. The experiment was repeated 1000 times for each parameter configuration.

2.3.8 Experiment Findings

Following the experiment, the authors found that the exit probability E_N increased with the increase in $E_a(0)$ for all strategies. Besides, the authors also found DMMD to be the least accurate and DC to be the most accurate. In contrast, DMMD was the fastest strategy and DC the slowest.

In addition, DC was also found to have poor scalability as its consensus time increased much faster than other strategies when the size of the swarm was increased from 20 to 100.

2.3.9 Limitations

Even though Valentini's decision-making strategies performed well in terms of exit probability and consensus time, they failed to account for Byzantine robots. Thus, the performance of these strategies in the presence of Byzantine robots was not studied.

The study conducted by Strobel et al. (2018) found Valentini's three strategies to fail to converge on the right opinion in the presence of around 4 or more Byzantine robots.

Byzantine robots are robots that lead to the Byzantine generals' problem. The Byzantine generals' problem is a situation where consensus needs to be achieved when some of the actors are unreliable (Lamport, Shostak and Pease, 1982). Thus, Byzantine robots can be defined as robots that are either malicious or malfunctioning.

This is a critical issue since, as the study by Higgins et al. (2009) shows, identity and authentication, and intrusion by foreign robots among others are major security challenges in a swarm robotic environment.

2.3.10 A Primer on Blockchain

A blockchain is a distributed ledger that was originally intended to be used to create a decentralized monetary system. However, with time, this technology has evolved to be used to create decentralized applications.

All participating nodes in a blockchain get a copy of the ledger. The ledger is a chain of blocks that contains information about transactions. Transactions are carried out using a pair of public and private keys (Nakamoto, 2009).

When a transaction needs to be sent to another party, the transactor addresses the transaction to the public key of the recipient and signs the transaction using their own private key. In order to be able to send money, the transactor must have already received the money. So, a successful transaction will have one or more input transactions that are addressed to the public key of the transactor.

The nodes in a blockchain can then verify the transaction by decrypting the signature of the transaction using the public key in the input transactions. Since signatures produced using the private key can only be decrypted by the corresponding public key, this acts as proof that the transactor owns the money that they sent.

The order of transactions matters to avoid double-spending. So, the transactions are added to a block. The blocks are then chained together using hashes, so the order becomes immutable. The process of producing blocks is called mining and is carried out by dedicated miners.

Miners compete with one another to produce the next block that would be added to the chain of blocks. This is an important process since all the nodes need to come to a consensus on which block should be added to their ledger next. The winner is determined by the use of a consensus algorithm with PoW being the most popular one at present.

The PoW algorithm requires miners to produce blocks whose hash value is less than a specified value. The difficulty of this process can be adjusted by lowering or raising this target value. Miners add nonces to the blocks before hashing them to try to produce the right block hash. When, finally, a node mines the right block, all the nodes agree to add this block to their chain.

The right hash value serves as proof that the miner did some work since producing the right hash requires continuous trial and error using CPU time, thus, giving the algorithm its name. This also makes the blockchain immutable since to mutate the order of blocks, the work done to produce them has to be repeated.

The immutability of the blockchain, its decentralized nature, and the ability to perform trustless transactions have made blockchain's application extend beyond monetary systems.

2.3.11 Proposed Architecture

Figure 2 Proposed architecture

2.4 Existing Applications of Blockchain in Swarm Robotics

Even though blockchain was originally intended to be used to decentralize monetary systems, its usefulness beyond financial technology has been increasingly realized. Blockchain's application in various disciplines such as document authenticity verification, insurance, the Internet of Things, the music industry, decentralized storage, and software defined networks have been explored (Crosby, 2016)(Krishnamohan *et al.*, 2020).

This section analyzes the use of blockchain in swarm robotics and in solving the Byzantine problem in particular. In order to analyze the existing works effectively, this research proposes a taxonomy as shown in Figure 3.

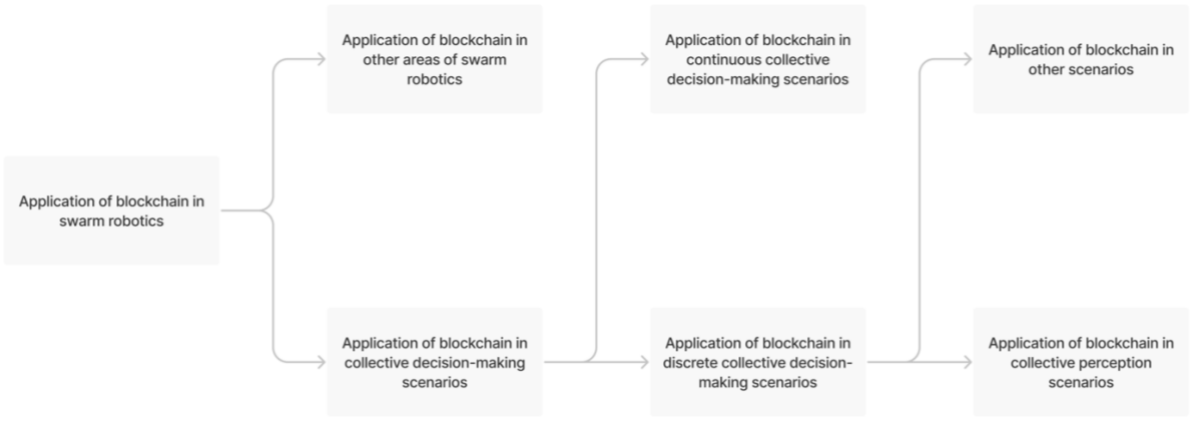


Figure 3 Taxonomy of applications of blockchain in swarm robotics

2.4.1 Application of Blockchain in Other Areas of Swarm Robotics

Karthik et al. (2020) used blockchain in swarm robotics to construct a two-dimensional structure. This research work used blockchain to carry out trustless communication and serves as an example of the use of blockchain in spatial organizing behaviors.

SwarmDAG, a novel protocol proposed by Tran et al. (2019), used blockchain to address challenges caused by network partitions in swarm robotics due to "navigational or communication" issues.

2.4.2 Application of Blockchain in Continuous Collective Decision-Making Scenarios

Strobel, Ferrer and Dorigo (2018) used a blockchain smart contract to reach a consensus in a continuous collective decision-making problem amidst the presence of Byzantine robots. The authors employed a smart contract to disregard outlying sensor readings, whereby they mitigated the security challenges caused by Byzantine robots (Strobel, Castelló Ferrer and Dorigo, 2020).

The superiority of the blockchain solution over classical solutions was proven by the authors by comparing the performance of the blockchain solution with classical solutions such as the Linear Consensus Protocol (LCP) and Weighted Mean Subsequence Reduced (W-MSR) algorithms in a simulated environment. The simulated experiment showed the blockchain solution to be able to tolerate more Byzantine robots than the state-of-the-art W-MSR algorithm.

2.4.3 Application of Blockchain in Discrete Collective Decision-Making Scenarios

Singh et al. (2020) used the Proof-of-Authority (PoA) algorithm as the consensus algorithm in their blockchain-based discrete decision-making strategy to solve the inherent challenges—such as excessive resource consumption—posed by using the de-facto PoW algorithm.

This strategy was demonstrated using a square grid that had three colors, viz. black, white, and gray. The middle of the square was gray in color whereas black and white colors were found at either end of the square grid.

The swarm of robots rested in the middle region initially before starting to explore the grid by moving randomly. If a robot sensed the gray color, no LED was lit. However, if the color white was detected, then the red LED was lit, while if black was detected, then the green LED was lit.

When a robot detected a color, while lighting its LED, it also sent a transaction to the blockchain. Only one transaction was allowed for a robot. The designers specified a threshold value in the smart contract and the color that reached the specified threshold value first was deemed the consensus color. The robots in the swarm periodically polled the smart contract to see if a consensus had been reached and if it had been, then they updated their opinion with the converged opinion.

The authors identified the use of the PoW algorithm, which is resource-intensive, as unsuitable for swarm robotics as the robots in a swarm are resource-constrained. Thus, they proposed the use of the PoA algorithm which used three validators to mine blocks.

However, it can be argued that this solution centralizes the task of mining in the three validators and, thus, nullifies some of the inherent benefits afforded by swarm robotics such as reliability and redundancy.

2.4.4 Application of Blockchain in Other Discrete Collective Decision-Making Scenarios

Nguyen et al. (2020) utilized blockchain to solve five types of best-of-n problems. The five types are:

- same quality same cost (Brambilla *et al.*, 2014)
- same quality different cost (Scheidler *et al.*, 2016)
- different quality same cost
- different quality different cost (Synergic)
- different quality different cost (Antagonistic)

In all of these scenarios, robots started from a rest area, moved around randomly while perceiving the quality of their opinion for a given time, and then returned back to the rest area to disseminate their opinion and its corresponding quality. The cost of the opinion was determined by the distance between the rest area and the area the robot explored.

When in the rest area, robots first checked if consensus had been reached by querying the smart contract. If consensus had been reached, then they entered the stop state. Otherwise, the robots estimated their influence factor. The influence factor if_{ij} of opinion i of robot j was given by:

$$if_{ij} = (w_1 \times q_i) + w_2(C \times c_{ij})$$

Equation 2.6

Where w_1 and w_2 were the positive weight for quality and cost ($0 \leq w_1; w_2 \leq 1$), q_i was the quality of the opinion i, c_{ij} was the exploration cost incurred by robot j to perceive opinion i, and C was a constant that denoted the maximum exploration time.

Once the influence factor was estimated, robots called the smart contract to submit their influence factor. Subsequently, robots called the smart contract to find the best opinion based on the influence factor and then voted for the best opinion. Once this was done, robots changed their opinion to the best opinion and began exploring again.

The authors showed that the blockchain approach provided a more accurate and faster performance than the non-blockchain approaches.

However, this solution also does not account for Byzantine robots and the use of the PoW algorithm means that this approach is ill-suited for resource-constrained robots. Besides, the generalizability of the proposed approach was not studied.

2.4.5 Application of Blockchain in the Collective Perception Scenario

Blockchain was applied to the collective perception scenario by Strobel et al. (2018) as the authors attempted to solve the Byzantine problem in the classical DMMD, DMVD, and DC strategies described in the **Consensus Achievement Strategies** section.

The authors deployed a private Ethereum network, and the mining difficulty was kept constant unlike in a typical blockchain network. An auxiliary geth node was used to publish the smart contract to the blockchain.

The robots in the swarm first registered themselves by sending a transaction to the blockchain. Once all the robots were registered, the auxiliary node stopped mining.

During registration, robots sent their public key to the smart contract and then listened to the events that were created during registration, which included the initial opinion of the robots, their block numbers, and the block hashes.

The robots followed the same routines as in the classical approach. However, during the dissemination state, instead of broadcasting their opinion to their neighbors, robots voted using the smart contract. Robots voted every 5 ticks (a second consists of 10 ticks), so the higher the quality of their opinion, the higher the number of votes was.

After voting, robots called the smart contract to execute the decision-making strategy. In the case of DMMD, the smart contract chose the opinions of two pseudorandom robots and returned the opinion of the majority. When DMVD was used, the opinion of a pseudorandom robot was returned.

When DC was used, robots passed both their opinion and the quality of their opinion to the smart contract. Then, the smart contract chose a pseudorandom robot and compared the quality of the opinion of the robot that called the smart contract with that of the pseudorandom robot and returned the opinion with the higher quality.

Following this, during the last 30 seconds of the dissemination state, the robots engaged in mining. This was to make sure the votes and the calls to get the best opinion were both executed

by the smart contract since these are sent as transactions and need to be mined to be included in the blockchain.

Strobel et al. (2018) used exogenous fault detection to detect Byzantine robots in the swarm (Christensen et al., 2009). A vote from a robot was ignored if it was based on an outdated opinion. An opinion was considered outdated if it had not been updated during the last 25 blocks. Robots were also allowed only a maximum of 50 votes during each dissemination state when DMMD and DMVD were used. In the case of DC, robots got only one vote. Moreover, votes were rejected if the blockchain versions were different.

Even though Strobel et al. (2018) managed to solve the Byzantine problem using blockchain, the swarm took longer to reach consensus in comparison to the classical approaches. This was because the blockchain used the PoW consensus algorithm which consumed extra time.

Additionally, the PoW algorithm is also resource-intensive and, hence, is not suitable to run on simple robotic devices.

Although Strobel et al. (2018) propounded a solution to the Byzantine problem that the classical decision-making strategies are vulnerable to, the Byzantine problem that stems from the PoW algorithm remained unsolved.

When the PoW algorithm is used, a node or a group of nodes with a hash rate in excess of 50% of the total hash rate of the network can successfully compromise the security of the network (Anita & Vijayalakshmi, 2019). This form of attack is popularly known as the 51% attack and the solution of Strobel et al. (2018) is susceptible to it.

In addition to this, the authors attempted to directly port the classical strategies to blockchain. In the classical strategies, one major phase was the dissemination of opinions of robots. This was important because robots can communicate their opinion only to their neighboring robots and the opinion does not get distributed across all robots. In contrast, peer-to-peer communication in blockchain ensures that information conveyed to one node gets distributed across all nodes. Thus, it can be argued that blockchain, by its own design, offers a better alternative to the dissemination phase of classical strategies.

Therefore, robots need to publish their opinion along with their quality to the smart contract only once. The robots can also spend less time in the dissemination state since communicating with a few robots is enough to send and receive blocks.

Thus, by developing a blockchain-native consensus strategy that exploits the inherent benefits of blockchain, the performance of the blockchain-based consensus strategy can be greatly improved.

2.5 A Review of Blockchain Consensus Algorithms

The 51%-attack problem resulting from the PoW algorithm was discussed above. Thus, the need for a different consensus algorithm can be realized. This section surveys other consensus algorithms and evaluates their suitability, or lack thereof, of being applied to swarm robotics.

2.5.1 Proof of Stake

This algorithm requires a miner to prove that it owns a certain number of coins and to lock a certain amount of currency as a stake in an escrow account to mine blocks (Ferdous *et al.*, 2020).

Miners will not have coins in the beginning, so no node can become a miner. This problem called the bootstrap problem is overcome by issuing pre-mined coins to miners. An alternative is to start with the PoW algorithm and then transition to the Proof-of-Stake algorithm. Besides, this algorithm is still susceptible to the 51%-attack problem (Ferdous *et al.*, 2020).

2.5.2 Proof of Authority

There are two major types of Proof-of-Authority (PoA) algorithms. One type of these algorithms requires a set of validators to be specified during the genesis of the blockchain and only these validators are allowed to mine blocks. Aura is an example of this type of algorithm.

This prevents robots from joining the swarm as miners after the blockchain is initiated. Besides, compromising the validators would compromise the entire swarm, which would nullify the benefit of reliability and redundancy offered by swarm robotics.

The other type allows miners to be dynamically added to the network through voting. The existing miners can vote to either include a new miner or exclude an existing miner. However, the algorithm does not specify on what basis miners should decide to vote in or vote out a miner. Clique is an example of this type of algorithm.

Thus, this algorithm cannot be used as it is in swarm robotics. To use this algorithm in swarm robotics, a protocol for voting should be established. Instead, a more straightforward approach would better fit swarm robotics.

2.5.3 Proof of Burn

This algorithm requires miners to burn their accumulated coins in order to mine blocks. Miners burn their coins by sending their coins to an address without a private key, effectively making the coins unusable (Ferdous *et al.*, 2020). The PoW algorithm has to be used initially until there are enough coins in the network (Ferdous *et al.*, 2020).

2.5.4 Proof of Capacity

This algorithm involves miners investing their disk space to mine blocks. This addresses the issue of energy wastage associated with the PoW algorithm (Mahmood & Wahab, 2020). Swarm robots will not have much disk space to use this algorithm.

2.5.5 Proof of Elapsed Time

This algorithm requires the miners to wait for a certain amount of time, which is randomly decided, before they are allowed to mine a block (Chen *et al.*, 2017). This will slow down the swarm and increase the consensus time.

2.5.6 Practical Byzantine Fault Tolerance

This algorithm uses a practical Byzantine state machine to achieve consensus (Liskov, 2010).

This algorithm can only be used in permissioned blockchains. Since any legitimate robot should be allowed to join a swarm continuously as a miner, this algorithm is not suitable. If a permissioned blockchain is used in a swarm, the swarm can be compromised by attacking the permissioned robots.

2.5.7 Proof of Research

This is a hybrid approach that combines the Proof-of-Stake algorithm with the Proof-of-BOINC algorithm. BOINC stands for Berkeley Open Infrastructure for Network Computing. This is a grid computing platform that allows scientific researchers to use the resources of personal computers all over the world. This requires a miner to share its resources with the BOINC network (Ferdous *et al.*, 2020).

Swarm robots have limited resources and cannot afford to share their resources with the BOINC network to mine blocks (Ferdous *et al.*, 2020).

2.5.8 Proof of Stake Velocity

This is similar to the Proof of Stake algorithm but, additionally, this algorithm incentivizes miners to stake their coins by rewarding miners who frequently stake their coins (Ferdous *et al.*, 2020).

This has the same limitations as the Proof-of-Stake algorithm.

2.5.9 Proof of Cooperation

This algorithm requires miners to have been authenticated by following a certain set of rules that require miners to complete some tasks. These tasks include resolving technical or management issues in the blockchain system or running a local node (Ferdous *et al.*, 2020).

It is impossible for robots in a swarm to invest their resources to solve issues in the blockchain network.

2.5.10 Proof of Importance

This extends the Proof-of-Stake algorithm by including more variables other than just the staked amount to pick the miner. This addresses the issue of coin hoarders having an advantage in Proof of Stake (Ferdous *et al.*, 2020). This has the same limitations as the Proof-of-Stake algorithm.

2.5.11 Stellar Consensus Protocol

Stellar Consensus Protocol (SCP) addresses the centralization issue in practical Byzantine Fault Tolerance (pBFT) by introducing quorum slices. This creates a federated Byzantine agreement system (Mazi`eres and Mazi`eres, no date). Each validator is allowed to have its own list of authorized validators. The validators recognized by a validator form a quorum slice. These quorum slices should intersect with other quorum slices forming a quorum. This makes the blockchain decentralized.

However, this results in a permissionless blockchain. This is not suitable for swarm robotics because this allows any validator to participate in the swarm. In addition, recent studies have called into question SCP's decentralized nature (Kim, Kwon and Kim, 2019).

2.6 Benchmarking and Evaluation

Strobel et al. (2018) and Valentini, Brambilla, et al. (2016) used the same metrics to evaluate their solutions to the collective perception scenario. These metrics are:

1. Exit probability
2. Consensus time

2.6.1 Exit Probability

Exit probability is the number of correct decisions divided by the total number of runs. In other words, when an experiment is repeated a certain number of times, this metric shows how many times the correct decision was arrived at. For instance, if the color of the majority of the tiles is black, the experiment is run 20 times, and 15 times the swarm concludes that the right color is black, then the exit probability is:

$$P(e) = \frac{15}{20}$$

Equation 2.7

2.6.2 Consensus Time

Consensus time denotes the average amount of time taken by a swarm to achieve the correct consensus. This metric is used to measure the speed of the algorithms that are used to solve the collective perception scenario. The research findings of Strobel et al. (2018) and Valentini, Brambilla, et al. (2016) serve as an excellent benchmark to compare the outcomes of future works and, hence, will be used to evaluate the outcomes of this research study as well.

2.7 Chapter Summary

Swarm robotics applies swarm intelligence to robotics to solve complex real-life problems. Existing research works in swarm robotics can be classified into two taxonomies, viz. methods taxonomy and behavior taxonomy. The behavior taxonomy includes the collective decision-making behavior, which can be used for task allocation and consensus achievement. Consensus achievement can be divided into discrete consensus achievement and continuous consensus achievement depending on if the consensus should be reached on a finite or infinite number of choices.

The collective perception scenario provides a generalized discrete consensus achievement challenge that can be used to test the effectiveness, speed, and generalizability of discrete consensus-achievement algorithms.

Existing solutions to the collective perception scenario are not immune to Byzantine robots. Studies have proved the efficacy of blockchain in addressing the issue of Byzantine robots in existing solutions to the collective perception scenario. However, such solutions suffer from poor speed and are still vulnerable to Byzantine robots as they offer no solution to the 51%-attack vulnerability endemic to the PoW consensus algorithm in blockchains.

An analysis of existing blockchain consensus algorithms revealed that no suitable consensus algorithm exists to address the Byzantine-robot issue completely in the current blockchain-based solutions.

Thus, this chapter established the need to design a new blockchain consensus algorithm to address the Byzantine-robot problem in swarm robotics in addition to improving the performance. Besides, this chapter also discussed the benchmarking and evaluation mechanisms to better evaluate the outcomes of this research work.

Chapter 3 Methodology

3.1 Chapter Overview

In this chapter, the methodologies used for research, design, development, project management, and evaluation are discussed in detail. Frameworks and models used in this research project and their relevance to this project are also examined extensively in this chapter.

3.2 Research Methodology

Philosophy	**Positivism** was chosen as the research philosophy among positivism, pragmatism, and interpretivism since this research work evaluated the performance of a swarm of robots using quantifiable metrics. Besides, this research work collected objective facts based on observations that were independent of the researcher. Hence, positivism was deemed the most appropriate philosophy.
Approach	This research used the **deductive** approach. This approach was used to test the validity of the theory that this research aimed to prove by running experiments and measuring the performance.
Strategy	The strategy chosen for this research was **prototyping** and **experimental** since a new consensus algorithm was prototyped first and the performance of the new consensus algorithm was tested by running experiments so that it could be compared with the performance of the existing strategies.
Choice	**Multi-method** was chosen as the research choice as the research involved multiple quantitative analyses such as analyzing the time taken for consensus and analyzing the exit probability using the data collected during the experiments.
Time Horizon	The time horizon chosen was **cross-sectional**. This was because the data was collected only at one point in time, i.e., during the experiments.

Table 3 Research methodology

The above methodology was used to determine the following attributes of the research:

Research Hypothesis: A blockchain consensus algorithm can be developed to be used in swarm robotics to address the 51%-attack issue seen with the Proof-of-Work (PoW) algorithm while allowing legitimate robots to authenticate themselves with the swarm as miners and providing a performance that is on par with the classical consensus-achievement strategies.

Prototype Input: A newly mined block.

Prototype Process: Verifying the authenticity of the robot that mined the block and ensuring that there is consensus.

Prototype Output: A blockchain with the newly mined block appended to it provided that the block passed the verification process.

Prototype Features:

1. Provides protection against the 51% attack.
2. Allows a swarm of robots to perform on par with the existing blockchain-based solutions.
3. Allows legitimate robots to authenticate themselves as miners with the swarm at any time.

3.3 Development Methodology

The agile methodology was used to manage the research project since, in research work, there can be a lot of uncertainties and ambiguities. A research environment fits a VUCA environment perfectly as you can observe the characteristics of VUCA such as volatility, uncertainty, complexity, and ambiguity in research work. Owing to this VUCA nature of research, methodologies such as waterfall, spiral, and prince2 were not considered.

3.4 Design Methodology

Function Oriented Design was used for analyzing and designing the solution since Go, the programming language that was used for development, supports functional programming better. Besides, since the research project aimed to build a consensus algorithm, a function-oriented design that decomposes a solution into multiple interacting modules was deemed better for the purpose.

3.5 Evaluation Methodology

The output of this research was evaluated using two metrics:

1. Exit probability (number of correct decisions over the total number of runs)
2. Consensus time (the average time the swarm takes to achieve the correct consensus)

The above metrics were chosen since the research works of both Strobel, Ferrer and Dorigo (2018) and Valentini, Brambilla, et al. (2016) used the same metrics. This allowed the

performance of the prototype produced by this research work to be compared with the previous research works effectively.

Besides, the Collective Perception scenario introduced by Valentini, Brambilla, et al. (2016) was used to run the experiments as this scenario is a generalized scenario that allows different strategies to be compared on an equal footing.

In order to allow comparisons, the experiments were carried out on environments with difficulty levels of 0.52, 0.56, 0.61, 0.67, 0.72, 0.79, 0.85, and 0.92 as this was the case with the experiments run by Strobel et al. Each experiment was repeated 10. This allowed the prototype of this research work to be compared with the work of Strobel, Ferrer and Dorigo (2018).

Moreover, the evaluation included the DMMD, DMVD, and DC consensus strategies so that the performance of the new consensus algorithm can be evaluated using all three strategies.

3.6 Project Management

The agile PRINCE2 methodology was used to manage the research project as this methodology enables strategic-level project management while allowing flexible and incremental deliveries.

3.6.1 Deliverables

Deliverable	**Date**
Project Proposal	17th February 2022
Literature Review	17th March 2022
Software Requirement Specification	14th April 2022
First Version of Prototype	12th May 2022
Interim Report with Improvised Version of Prototype	2nd June 2022
Testing and Evaluation	23rd June 2022
Draft Report	7th July 2022
Final Thesis and Prototype	28th July 2022

Table 4 Project deliverables

3.6.2 Gantt Chart

The Gantt Chart can be found in the Appendix section under **Gantt Chart**.

3.6.3 Resource Requirements

Resource requirements can be found in the Appendix section under **Resource Requirements**.

3.6.4 Risk Management

Risk Item	Severity	Frequency	Risk Exposure	Mitigation Plan
Inability to extend Ethereum with a new consensus algorithm.	5	1	5	Explore the feasibility of using other blockchain platforms such as Hyperledger.
Inability to interface Ethereum with ARGoS Simulator.	5	1	5	Explore the possibility of interfacing other blockchain platforms with ARGoS.
Having to implement existing strategies in code to run the experiments.	3	2	6	Allocate buffer time to allow the implementation of existing strategies should the code be not available.
ARGoS failing to run on the laptop to be used for the research.	5	1	5	Find a backup laptop or use virtualization if the primary one fails to run ARGoS.
ARGoS having a steep learning curve that makes completing the project within the specified timeline tough.	3	1	3	Allocate buffer time to learn to use ARGoS.

Table 5 Risk management

3.7 Chapter Summary

This chapter discussed the research methodology used under the topics of philosophy, approach, strategy, choice, and time horizon. Then, the development methodology was discussed followed by the design methodology. Next, this chapter discussed the evaluation methodology used before expounding the project-management methodology.

Chapter 4 Software Requirements Specification

4.1 Chapter Overview

This chapter dwells on gathering requirements for the research project. The chapter first gives a rich picture of the system and the environment it operates on, followed by the stakeholder analysis using Saunder's Research Onion model. Then, stakeholder roles are discussed, following which the requirement elicitation methodologies utilized are discussed in detail. Further, this chapter also discusses the findings of different requirement elicitation methodologies and summarizes the findings. This chapter also includes a context diagram and a use-case diagram, which are followed by a description of the use cases. Finally, this chapter discusses the functional and non-functional requirements of this project.

4.2 Rich Picture of the System

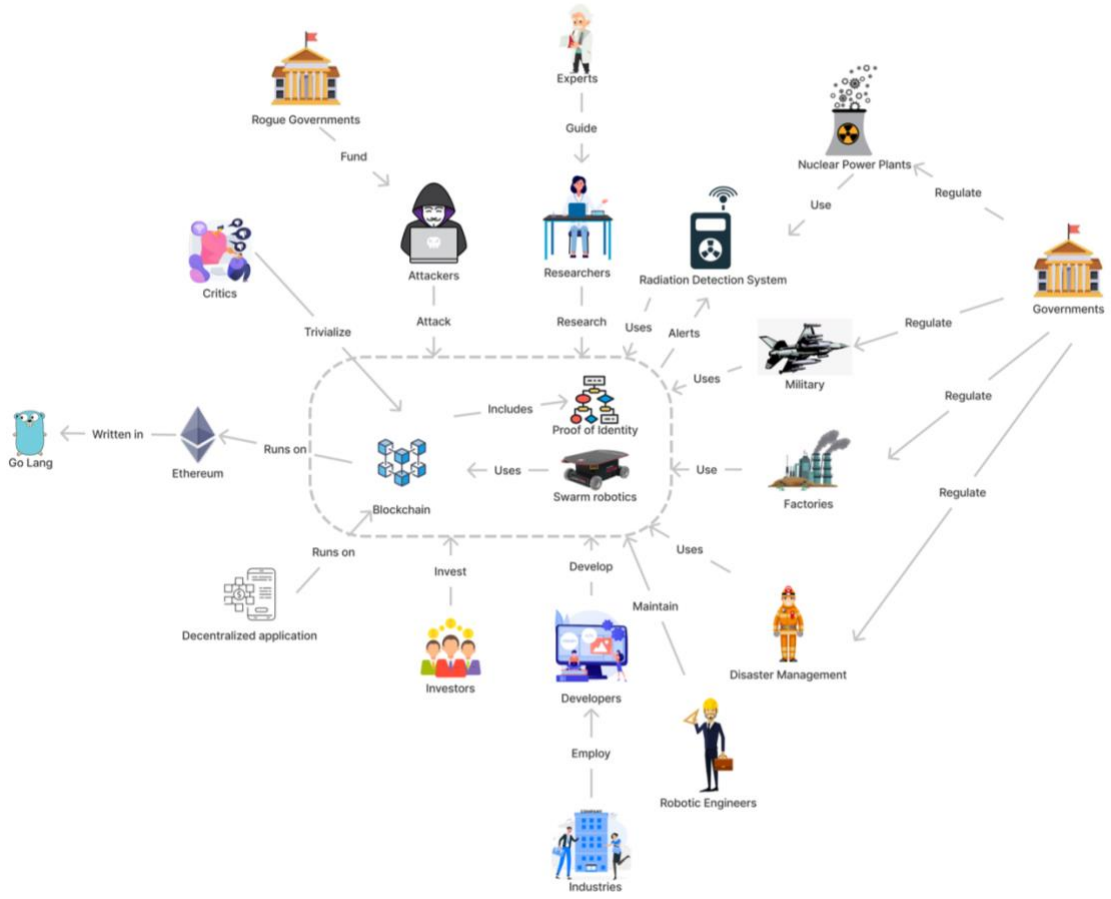


Figure 4 A rich picture of the system

4.3 Stakeholder Analysis

4.3.1 Stakeholder Onion Model

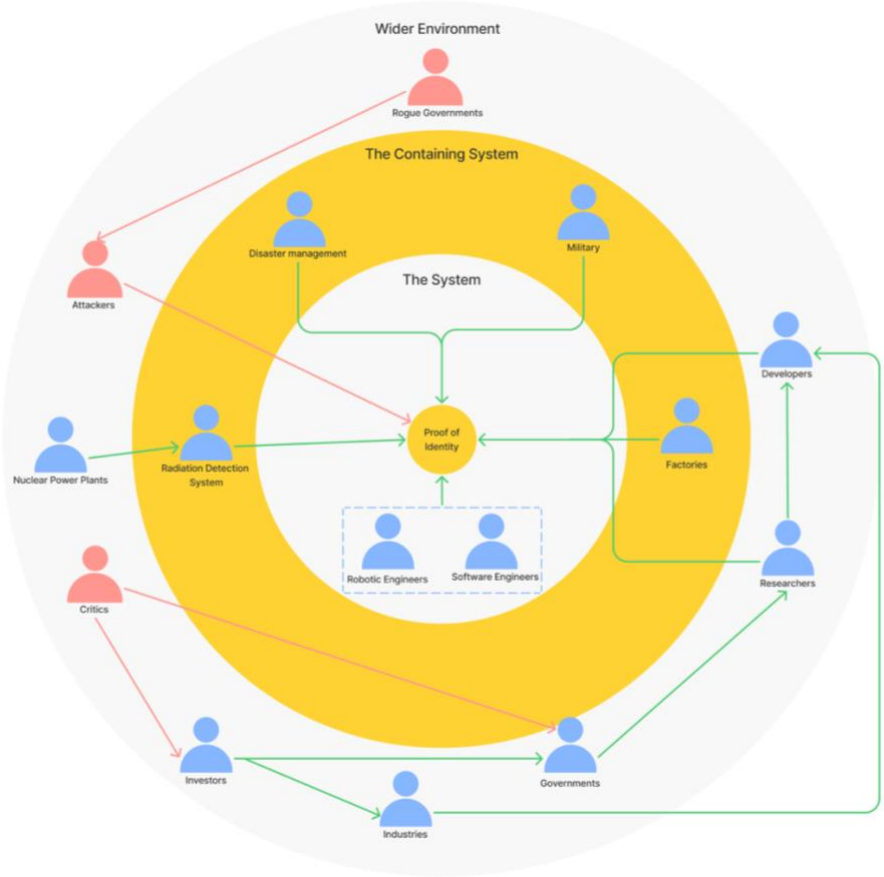


Figure 5 Stakeholder onion model

4.3.2 Stakeholder Viewpoints

Stakeholder	Role	Description
System Stakeholders		
Robotic Engineers	System maintenance	They are responsible for the maintenance and upkeep of the hardware and firmware.
Software Engineers	Software maintenance	They are responsible for the maintenance of the consensus algorithm and the smart contract running on blockchain.
Containing System Stakeholders		
Military	Functional beneficiary	Swarm robotics can be used in the military for aerial reconnaissance skirting air defense systems and carrying out dangerous operations.

Disaster Management		Disaster management can use swarm robotics to detect smoke and fire, map regions affected by disaster, and reach affected regions that could be hostile to humans.
Factories		Factories can use swarm robotics to detect leakages, and fire, and to monitor the factory environment.
Nuclear Power Plants		Swarm robotics can be used to detect radiation leakage.
Wide Environment Stakeholders		
Developers	Engineering Staff	They develop the smart contract and the consensus algorithm required for the functioning of the swarm.
Governments	Functional Beneficiary	They fund swarm robotics research and development and offer regulatory guidelines.
Rogue Governments	Negative stakeholders	Rogue governments can fund hackers and cyber criminals to attack and compromise robotic swarms.
Attackers		Attackers can try to compromise robot swarms by carrying out attacks against the blockchain network, the smart contract, or the robots themselves.
Critics		Critics of swarm robotics and blockchain can propagate negative sentiments against these technologies which can impede their adoption and funding.
Investors	Financial beneficiary	They invest in blockchain and swarm robotics, driving research and development.
Researchers	Advisory	They carry out research into improving blockchain and swarm robotics, decreasing the cost of adoption, and making them more secure and immune to security threats.
Industries	Financial beneficiary	Industries involved in blockchain, and swarm robotics development make money by selling blockchain and swarm robotics solutions to the market.

Table 6 Stakeholder viewpoints

4.4 Requirement Elicitation Methodology

Various methodologies can be utilized to garner requirements for a research project. This section discusses in detail the various requirement elicitation methodologies used and the rationale behind the decision to choose them. On this account, literature review, prototyping,

self-evaluation, formal interviews, and individual brainstorming were chosen as the requirement elicitation methodologies.

4.4.1 Literature Review

A literature review on the disciplines of swarm robotics and blockchain could demonstrate the limitations in current technologies and the gaps that need to be filled to increase the adoption of these technologies. An adequate amount of research has already been carried out on the application of blockchain in swarm robotics, especially in the discrete consensus achievement problem. By analyzing the existing literature, one could identify what has been accomplished, what needs to be accomplished, and what needs to be improved. Thus, literature review was chosen as a methodology to elicit requirements.

4.4.2 Prototyping

Since this research project used prototyping and experimental research strategies, prototyping was also chosen as a requirement elicitation methodology. By prototyping, experimenting, and analyzing the outcomes iteratively, shortcomings in the prototype and opportunities to improve can be discovered. Such findings can become requirements for the succeeding iteration.

4.4.3 Self-Evaluation

Evaluating existing solutions could help identify their limitations and this could be used to identify the requirements of this research project. Since there are sufficient blockchain consensus algorithms available and there is a rich body of scholarship on the use of blockchain in swarm robotics, evaluating them was a potent requirement elicitation methodology.

4.4.4 Individual Brainstorming

Individual brainstorming could aid in determining requirements that cannot be established through literature review and self-evaluation. Even though there is an appreciable body of knowledge about the application of blockchain in swarm robotics, and the various blockchain consensus algorithms, the literature on the use of different blockchain consensus algorithms in swarm robotics, specifically in the discrete consensus-achievement problem space is scarce. Thus, brainstorming could help discover new requirements for a blockchain consensus algorithm.

4.4.5 Formal Interviews

Requirements could be gathered by interviewing experts both in the subject domain and the technical domain. This helps in understanding the domains better and allows the identification of challenges preemptively. Their expertise and experience in their respective fields would be helpful in identifying requirements that could not be gleaned from other elicitation strategies.

4.5 Discussion of Findings of Each Requirement Elicitation Methodology

4.5.1 Literature Review

Findings	Citations
Robotic swarms are vulnerable to individual robots being physically captured and tampered with since robots in swarms are often deployed in remote regions that are not directly under the control of the owners of the swarms. Robots captured and tampered with can then be used to compromise the entire swarm, which can have far-reaching financial and safety consequences. So, swarms should be immune to individual robots being physically compromised.	(Higgins et al., 2009)
Having separate miners would mean that a swarm cannot be distributed over a large area, and, thus, robots should also act as miners.	(Singh *et al.*, 2020)
The blockchain consensus algorithm should be simple enough to run on small, powerless robots.	(Strobel et al., 2018)
The blockchain should be able to reach consensus faster and be resistant to the 51% attack.	(Strobel et al., 2018)

Table 7 Requirements elicited from literature review

4.5.2 Prototyping

Criteria	Findings
The nature of miners	The robots should be able to function as miners since sedentary miners can nullify the inherent benefits of swarm robotics such as mobility and distribution.
The type of blockchain to be used	A permissioned blockchain will allow faster consensus without consuming too much CPU time. However, since a permissioned blockchain does not allow new miners to be added dynamically, that

	could compromise on the distributed nature of swarm robotics. Hence, the blockchain should be dynamically permissioned so that new authorized miners can join the blockchain network anytime.
Blockchain branches	Too many branches were also identified as a problem during prototyping. When mining becomes easy, the probability of multiple simultaneous mining of blocks increases creating branches in the blockchain. The new blockchain consensus algorithm should have in-built mechanisms to either avoid or resolve the issue of too many branches.
Block structure	The consensus algorithm should be compatible with the existing Ethereum block structure to ensure this algorithm can be plugged into the existing codebase seamlessly.
Deploying miners	There should be a way by which the owners of a swarm can easily deploy new miners.

Table 8 Requirements elicited from prototyping

4.5.3 Self-Evaluation

A self-evaluation of the existing blockchain consensus algorithms was carried out against the following criteria:

- Whether or not the blockchain consensus algorithm creates a permissioned blockchain (Permissioned blockchain).
- Whether or not the consensus algorithm allows new miners to join the blockchain network (Allows new miners).
- Whether or not the consensus algorithm is simple enough to run on powerless devices (Simple consensus algorithm).
- Whether or not the consensus algorithm has mechanisms to address the issues of branches or to avoid them altogether (Branches are non-existent/addressed).
- Whether or not the consensus time is short enough (Shorter consensus time).
- Whether or not the consensus algorithm is resistant to the 51% attack in a network that contains only a few less-performant robots (Immune to 51%-attack).

Consensus Algorithms \ Criteria	Permissioned Blockchain	Allows New Miners	Simple Consensus Algorithm	Branches are Non-Existent/Addressed	Shorter Consensus Time	Immune to 51% Attack
Proof of Stake		✓	✓	✓	✓	
Proof of Authority	✓		✓	✓	✓	✓
Proof of Burn		✓	✓	✓	✓	
Proof of Capacity		✓		✓	✓	
Proof of Elapsed Time		✓		✓		
Practical Byzantine Fault Tolerance	✓		✓	✓		✓
Proof of Research		✓		✓	✓	
Proof of Stake Velocity		✓	✓	✓	✓	
Proof of Cooperation		✓		✓		
Proof of Importance		✓	✓	✓	✓	
Proof Work		✓		✓		

Table 9 Self-evaluation of existing consensus algorithms

A self-evaluation of the existing blockchain consensus algorithms revealed that a permissioned blockchain is required to address the 51%-attack vulnerability in swarm robotics. In addition, neither of the two permissioned consensus algorithms allows new miners to join, which could violate the principles of swarm robotics. Thus, the new blockchain consensus algorithm should be dynamically permissioned. Moreover, all the consensus algorithms either address the issue of multiple branches or ensure it does not occur. Additionally, most of them offer a shorter consensus time as well.

4.5.4 Individual Brainstorming

Through individual brainstorming, it was found that the robots should also function as miners since it is impractical to have dedicated mining robots in a swarm. At the same time, it was

found that only authorized robots should be able to mine, and new miners should be able to be introduced at any point in time.

On top of that, the consensus algorithm should be able to run on simple robots such as kilobots and e-pucks, which are not equipped with powerful CPUs. The consensus algorithm should also consume less energy since the robots might not have powerful batteries.

Further, the algorithm should be able to achieve consensus quickly so as not to affect the consensus time of the swarm. The algorithm should also be resistant to the 51% attack and owners of the swarm should be able to deploy new miners easily.

4.5.5 Formal Interviews

Four experts in swarm robotics and three experts in blockchain were interviewed to gather requirements. The four experts in swarm robotics included a founder and Chief Executive Officer (CEO) of a swarm robotics company, a post-doctoral researcher in swarm robotics, a Ph.D. candidate researching swarm robotics, and an engineer with a master's degree in artificial intelligence. The three experts in blockchain consisted of two blockchain developers and a co-founder of a blockchain startup.

Thematic analysis was carried out on the responses received during the interviews and the findings are furnished below.

	Theme	**Analysis and Findings**
1	Impact of consensus time	Interviewees largely agreed that consensus time is critical for the application of blockchain in different fields but averred that it can be more critical in certain fields than in others.
2	The threat of the 51% attack	The views of the interviewees were divided on the significance of the 51%-attack threat. One interviewee claimed that is less of a concern if a proper Decentralized Autonomous Organization (DAO) is in place, while the rest claimed that depending on the consensus algorithm and the hash rate of the network, this could be a big threat.
3	Lower mining difficulty	Interviewees agreed that mining should be difficult and if it is not, mechanisms should be in place to resolve multiple branches in a blockchain.

4	The risk of robots being physically captured and tempered with	While one interviewee said that it is impractical for an attacker to capture a robot, understand its functions, reprogram it and redeploy it, others agreed that this is a relevant threat, especially when they are deployed in remote locations. One of these interviewees also stated that this threat is negligible in highly controlled environments.
5	Byzantine robots	While most interviewees agreed that if there are enough Byzantine robots, the swarm could be affected, one interviewee was of the opinion that this threat is not usually considered in real-world deployments as the researchers and engineers are more interested in "getting something to work". However, the interviewee admitted that this could “pose a major threat” and that vulnerability to Byzantine robots may “have consequences for legal decisions”.
6	51% attack being a type of Byzantine robot problem	All of the interviewees agreed that a Byzantine robot could also pose a threat at the consensus level and thus, making the 51%-attack problem a Byzantine robot problem as well.
7	Dynamic deployment of robots to the swarm	Interviewees unanimously agreed that robots should be able to be dynamically deployed to a swarm as it is difficult to gauge the number of robots required in advance and failing robots will have to be replaced by new robots in the swarm.

Table 10 Thematic analysis of the interview responses

4.6 Summary of Findings

	Findings	**Literature Review**	**Prototyping**	**Self-Evaluation**	**Individual Brainstorming**	**Formal Interviews**
1	Robotic swarms should be immune to attacks that involve physically capturing and tampering with individual robots.	✓				✓

2	Robots should also function as miners.	✓	✓		✓	
3	The blockchain to be used should be a permissioned blockchain.		✓	✓	✓	
4	Miners should be able to be added to the blockchain network dynamically (dynamically permissioned blockchain).		✓	✓	✓	
5	The blockchain consensus algorithm should be simple enough to run on powerless CPUs.	✓	✓	✓	✓	
6	The blockchain consensus algorithm should avoid too many branches.		✓	✓		✓
7	The blockchain consensus algorithm should be able to achieve consensus within a short time.	✓		✓	✓	✓
8	The blockchain consensus algorithm should be resistant to the 51% attack.	✓		✓	✓	✓
9	The blockchain consensus algorithm should be compatible with the existing Ethereum block structure.		✓			
10	Swarm owners should be able to deploy new miners in a simple way.		✓		✓	

Table 11 Summary of findings

4.7 Context Diagram

There are four main users of the system, namely the swarm controller, swarm administrators, and e-puck robots in the ARGoS simulator. The swarm controller signs miners and distributes its public key, while the swarm administrators deploy miners to the blockchain network. The e-puck robots publish their opinions to the blockchain, validate, and mine blocks.

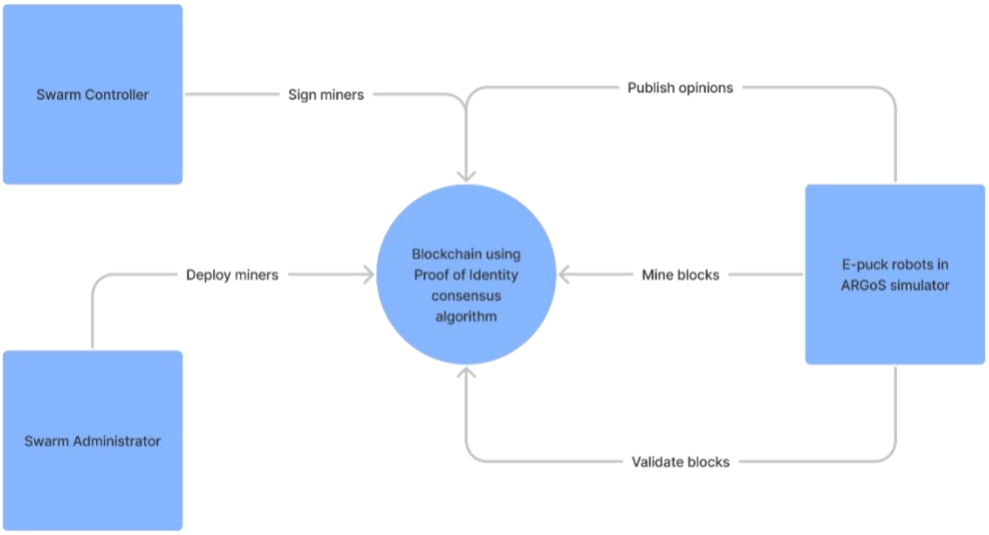


Figure 6 Context diagram

4.8 Use Case Diagram

The use case diagram consists of three actors, viz. swarm administrator, swarm controller, and robot. Swarm administrator generates public-private key pair and deploys a new robot. The swarm controller encrypts the addresses of new robots and distributes the public key to the robots in the swarm. The robot generates, validates, and adds blocks to the blockchain.

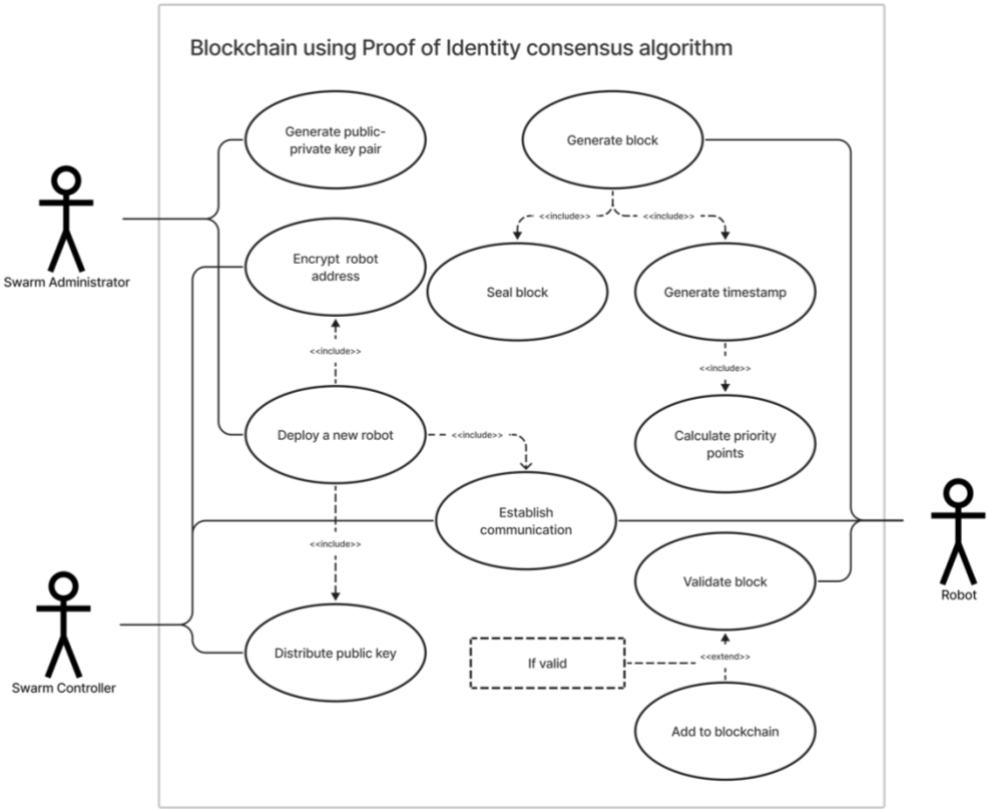


Figure 7 Use case diagram

4.9 Use Case Descriptions

Use Case Name	**Generate public-private key pair**
Description	A swarm administrator generates a public-private key pair for the swarm controller.
Participating Actors	Swarm Administrator
Pre-conditions	The swarm controller is up and running and a new swarm is about to be deployed.
Extended Use Cases	N/A
Included Use Cases	N/A
Main Flow	1. The swarm administrator gets a request to deploy a new swarm. 2. The swarm administrator generates a private-public key pair. 3. The private key is stored safely in the swarm controller.
Alternative Flow	N/A
Exception Flows	N/A
Post-conditions	A private key is stored in the swarm controller safely and the corresponding public key is available.

Table 12 Description of the "Generate private-public key pair" use case

Use Case Name	**Deploy a new robot**
Description	The swarm administrator deploys a new robot into a swarm.
Participating Actors	Swarm Administrator
Pre-conditions	A swarm has been already deployed and a robot is within the range of the swarm controller.
Extended Use Cases	N/A
Included Use Cases	• Encrypt robot address • Distribute public key • Establish communication
Main Flow	1. The swarm administrator triggers the robot to establish communication with the swarm controller. 2. The robot's address is encrypted using the private key. 3. The public key is sent to the robot. 4. The connection is terminated, and the robot joins the swarm.

Alternative Flow	N/A
Exception Flows	N/A
Post-conditions	A new robot is deployed into the swarm.

Table 13 Description of the "Deploy a new robot" use case

Use Case Name	**Encrypt robot address**
Description	The swarm controller encrypts the address of a new mining robot with the private key.
Participating Actors	Swarm Controller
Pre-conditions	A private key is stored in the swarm controller safely and the corresponding public key is available.
Extended Use Cases	N/A
Included Use Cases	N/A
Main Flow	1. The robot establishes communication with the swarm controller. 2. The swarm controller obtains the address of the mining robot. 3. The swarm controller encrypts the address with the private key. 4. The swarm controller sends the encrypted address (signature) to the mining robot.
Alternative Flow	N/A
Exception Flows	N/A
Post-conditions	The robot has its signature in its system.

Table 14 Description of the "Encrypt robot address" use case

Use Case Name	**Distribute public key**
Description	The swarm controller distributes its public key to the robots.
Participating Actors	Swarm Controller
Pre-conditions	A private key is stored in the swarm controller safely and the corresponding public key is available.
Extended Use Cases	N/A
Included Use Cases	N/A
Main Flow	1. The robot establishes communication with the swarm controller.

	2. The swarm controller sends its public key to the robot. 3. The robot stores the public key in its memory. 4. The connection is terminated.
Alternative Flow	N/A
Exception Flows	N/A
Post-conditions	The robot has the public key of the swarm controller.

Table 15 Description of the "Distribute public key" use case

Use Case Name	**Establish communication**
Description	The swarm controller and robot establish communication between them.
Participating Actors	Swarm Controller, Robot
Pre-conditions	The robot and the swarm controller are within each other's range.
Extended Use Cases	N/A
Included Use Cases	N/A
Main Flow	1. The robot initiates communication with the swarm controller. 2. Robot and the swarm controller exchange messages.
Alternative Flow	N/A
Exception Flows	N/A
Post-conditions	The robot and the swarm controller can communicate with each other.

Table 16 Description of the "Establish communication" use case

Use Case Name	**Generate block**
Description	The robot generates a block and seals it.
Participating Actors	Robot
Pre-conditions	The robot has received unverified transactions. The robot has its signature
Extended Use Cases	N/A
Included Use Cases	• Attach signature to block • Generate timestamp
Main Flow	1. The robot takes transactions from the pool of unverified transactions. 2. The robot verifies the transactions and packs them into a block.

	3. The robot adds its signature to the block. 4. The robot generates a timestamp and adds it to the block. 5. The robot hashes the block. 6. The robot publishes the block.
Alternative Flow	N/A
Exception Flows	N/A
Post-conditions	The robot has published a block.

Table 17 Description of the "Generate block" use case

Use Case Name	**Generate timestamp**
Description	The mining robot generates a timestamp based on certain parameters and adds it to the blockchain so that it can be used to avoid blockchain branches.
Participating Actors	Robot
Pre-conditions	A block has been generated.
Extended Use Cases	N/A
Included Use Cases	Calculate priority points
Main Flow	1. The robot gets the current time. 2. The robot calculates the priority points. 3. The robot generates a timestamp by adding the priority points to the current time.
Alternative Flow	N/A
Exception Flows	N/A
Post-conditions	The block has a timestamp.

Table 18 Description of the "Generate timestamp" use case

Use Case Name	**Calculate priority points**
Description	The robot calculates its priority points.
Participating Actors	Robot
Pre-conditions	A block has been generated.
Extended Use Cases	N/A
Included Use Cases	N/A
Main Flow	1. The robot finds how many of the previous n number of blocks were mined by itself.

	2. The robot calculates the offset value based on the number and the position of the blocks that were mined by itself. 3. The robot adds a random value to the offset value to produce the priority points.
Alternative Flow	N/A
Exception Flows	N/A
Post-conditions	The block has priority points.

Table 19 Description of the "Calculate priority points" use case

Use Case Name	**Seal block**
Description	The mining robot attaches its signature to the block and seals it.
Participating Actors	Robot
Pre-conditions	The robot has its signature in its system. The robot has generated a block. The robot has generated the timestamp.
Extended Use Cases	N/A
Included Use Cases	N/A
Main Flow	1. The robot fetches its signature from its memory. 2. The robot attaches its signature to the block. 3. The robot adds the block to its blockchain. 4. The robot publishes the block.
Alternative Flow	N/A
Exception Flows	N/A
Post-conditions	The block has the robot's signature.

Table 20 Description of the "Attach signature to the block" use case

Use Case Name	**Validate block**
Description	Other robots in the swarm validate a newly mined block by validating the signature.
Participating Actors	Robot
Pre-conditions	The robot has the public key. A new block has been mined by another robot.
Extended Use Cases	Add to the blockchain (If the block is valid)
Included Use Cases	N/A

Main Flow	1. The robot receives the new block 2. The robot verifies the transactions in the new block. 3. The robot decrypts the signature of the block with the public key and obtains the address. 4. The robot verifies the address of the robot that mined the block.
Alternative Flow	N/A
Exception Flows	3.1. The decryption fails. 3.2. The robot rejects the block. 4.1 The address verification fails. 4.2 The robot rejects the block.
Post-conditions	The new block is verified.

Table 21 Description of the "Validate block signature" use case

Use Case Name	**Add to blockchain**
Description	Robot adds a verified block to its blockchain
Participating Actors	Robot
Pre-conditions	There is a verified block.
Extended Use Cases	N/A
Included Use Cases	N/A
Main Flow	1. The robot appends the block to its blockchain. 2. If there is more than one block referring to the same previous block, the robot picks the block with the earliest timestamp. 3. The robot rejects all other blocks.
Alternative Flow	N/A
Exception Flows	N/A
Post-conditions	The new block is added to the blockchain.

Table 22 Description of the "Add to blockchain" use case

4.10 Functional Requirements with Priorities

The MoSCoW technique was used to define the priority of the system requirements based on their importance.

Priority Level	Description
Must Have (M)	Requirements belonging to this priority level form the core functions of the prototype and are mandatory to be implemented.
Should Have (S)	These are important requirements that are not a vital part of the prototype but will add significant value.
Could Have (C)	These are desirable requirements that are optional and not considered important for the completion of the prototype.
Will Not Have (W)	These are the requirements that will be absent from the prototype as they are not important at this time.

Table 23 Priority levels based on the MosCoW technique

ID	Functional Requirement	Priority	Relevant Use Case
FR1	Swarm administrators must be able to generate a public-private key pair.	M	Generate private-public key pair
FR2	The swarm controller must be able to store the private key.	M	Generate private-public key pair
FR3	The swarm controller must be able to encrypt the address of mining robots with the private key to produce signatures.	M	Encrypt robot address
FR4	The swarm controller must be able to send the signature to the robot.	M	Encrypt robot address
FR5	The swarm controller must be able to distribute the public key to the robots.	M	Generate private-public key pair
FR6	The robots must be able to store the public key and signature in their memory.	M	Generate private-public key pair, Encrypt robot address
FR7	The robots must be able to attach their signature to the blocks they mine.	M	Attach signature to the block
FR8	The robots must be able to generate a timestamp for their block.	M	Generate timestamp
FR9	The robots must be able to validate blocks using the public key.	M	Validate block signature
FR10	The consensus algorithm should avoid or reduce the number of branches created in the blockchain.	S	Generate timestamp

Table 24 Functional requirements

4.11 Non-Functional Requirements

ID	Requirement	Description	Priority
NFR1	Performance	The blockchain consensus algorithm must be able to achieve consensus quickly so that the swarm can reach its consensus faster.	M
NFR2	Simplicity	The blockchain consensus algorithm must not be complex and should be simple enough to run on low-powered, light robots.	M
NFR3	Security	The consensus algorithm must be immune to the 51% attack.	M
NFR4	Usability	The swarm administrators who use the swarm controller should be able to generate a public-private key pair easily.	S
NFR5	Compatibility	The new consensus algorithm should be compatible with the Ethereum block structure.	M

Table 25 Non-functional requirements

4.12 Chapter Summary

This chapter produced a rich picture of the environment the system is expected to operate on and used it to identify the stakeholders. Then, the stakeholders were categorized using Saunder's Research Onion and their roles were analyzed. Next, the requirement elicitation methodologies that were chosen were examined and justified. Subsequently, the gathered requirements and the findings inferred were discussed. Besides, this chapter presented a context diagram and a use-case diagram. Finally, the use-cases were described in detail, followed by the appraisal of functional and non-functional requirements.

Chapter 5 Social, Legal, Ethical, and Professional Concerns

5.1 Chapter Overview

This chapter discusses the social, legal, ethical, and professional issues and concerns that arose during the research project and how they were resolved or mitigated.

5.2 Social Issues and Mitigation

- All interviewees who were interviewed both for requirement gathering and project evaluation were clearly informed about the research work, how their responses would be utilized, and that an anonymized summary of their responses would be included in the thesis.
- Interviewees were assured that their identity would be kept confidential and only their names and designations were collected to be used during analysis.
- The responses of the interviewees were not directly used as they were in the thesis. Thematic analysis was performed using the collected responses and only the output of the analysis was used in the thesis.

5.3 Legal Issues and Mitigation

- All the programming languages and their compilers used in this project use open-source licenses. Besides, the code bases used in this project also use open-source licenses.
- The code written for this project is also open source and licensed under Apache License 2.0.
- The personal data of the interviewees and their privacy were protected. Responses were collected using Microsoft Office Forms and the responses were stored only in the cloud. The data was not downloaded or retrieved to any other medium.
- No personally identifiable information of the interviewees is published in the thesis unless explicit consent was given.

5.4 Ethical Issues and Mitigation

- The code bases used in this project are open source and freely accessible. When modifications were needed, existing repositories in GitHub were forked and the commit history of the original authors was left intact so as to maintain a reference to the original work and to credit the original authors.

- No part of this research project involved plagiarism and the works of others used are cited and referenced. The thesis is also free of plagiarism and involves no fabrication, exaggeration, or falsification.

5.5 Professional Issues and Mitigation

- This project only used open-source tools and software and pirated or illegally obtained software was not used.
- The data obtained from the experiments are accurate and not tampered with.
- Industry standards and best practices were used throughout the project lifecycle.
- The prototype was developed in a secure environment that was password-protected and up to date with the latest security updates.
- The code produced is securely stored in GitHub, a cloud-based code repository that is popular for open-source projects.

5.6 Chapter Summary

This chapter discussed the social, legal, ethical, and professional concerns in the research project and how they were avoided or mitigated. Concerns regarding the privacy of the interviewees, data protection, software licenses, and plagiarism were examined in this chapter, and preventive and mitigative measures taken were also presented.

Chapter 6 System Architecture and Design

6.1 Chapter Overview

This chapter discusses the architecture and design of the benchmarking tool and the design of the Proof-of-Identity (PoI) algorithm in detail. Besides, this chapter also discusses the rationale behind the design decisions that were taken. The requirements, including both functional and non-functional, for the design and architecture were gathered through literature review, prototyping, self-evaluation, individual brainstorming, and formal interviews.

6.2 Design Goals

Requirement	Description
Performance	The blockchain consensus algorithm should not take too long to converge as it would affect the time the swarm takes to achieve consensus. The default Proof-of-Work (PoW) consensus algorithm typically takes a significant amount of time to achieve consensus. Even though this time could be reduced by reducing the mining difficulty of the algorithm, this compromises the security of the algorithm, making it even more vulnerable to 51% attacks. Thus, the blockchain consensus algorithm to be developed should be able to achieve consensus faster without compromising on security.
Simplicity	Since the e-puck robots that would be used in this experiment are low-powered, less-powerful devices, the consensus algorithm should be simple enough to run on such devices without being CPU intensive and consuming a lot of power.
Security	The consensus algorithm should provide protection from the 51% attack. In order to prevent the 51% attack, the consensus algorithm should make sure rogue miners with powerful CPUs or GPUs do not get to mine blocks successfully.
Usability	Swarm administrators should be able to deploy miners that use the new consensus algorithm easily. In addition, researchers who use the collective perception scenario to benchmark solutions should be able to perform benchmarking in a simpler manner. The existing benchmarking solution is based on a Command-Line Interface (CLI) and stores experiment results in

	an unstructured file. To improve the user experience, a Graphical User Interface (GUI) should be developed, and the experiment results should be persisted in a database.

Table 26 The design goals

6.3 System Architecture

6.3.1 The Architecture of the Benchmarking Tool

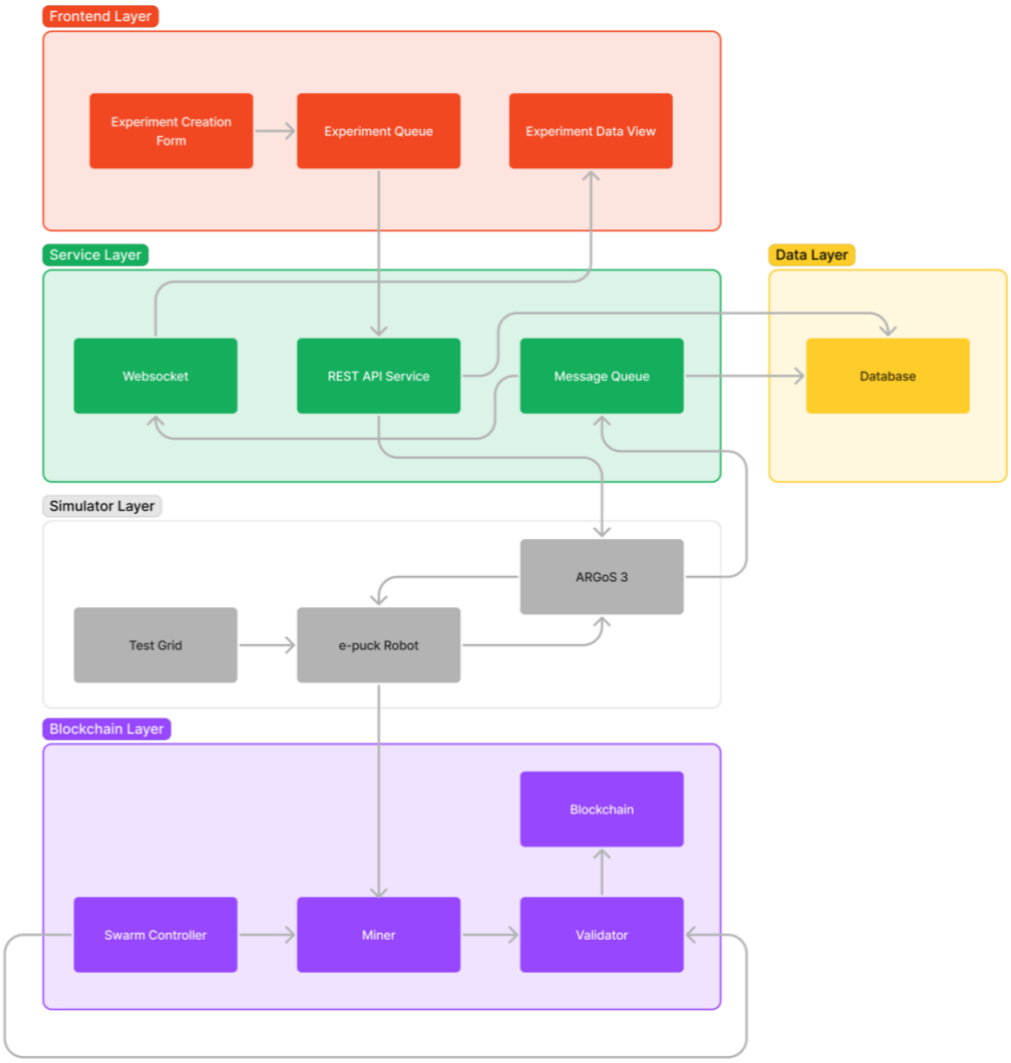


Figure 8 The layered architecture of the benchmarking tool

The layered architecture of the prototype consists of the frontend layer, service layer, simulator layer, and blockchain layer. The frontend layer provides the user of this prototype with a user interface to interact with the prototype. The service layer sits in between the frontend layer and the simulator layer and provides the necessary APIs to the frontend layer to communicate with the simulator layer. The simulator layer interacts with the blockchain layer to solve the

collective perception scenario using the smart contract deployed in the blockchain. The forthcoming section discusses these layers and the modules belonging to them elaborately.

6.3.1.1 The Frontend Layer

This layer consists of the Graphical User Interface (GUI) that a user will be using to interact with the prototype. It consists of the following modules:

- Experiment Creation Form—This is a form that will allow a user to configure the parameters of the experiment such as the number of robots, the decision rule to be used, the percentage of black and white tiles, the number of Byzantine robots and the approach to be used.
- Experiment Queue—Since, to benchmark different solutions, a user will need to run experiments in batches, experiments created using the Experiment Creation Form are added to this queue. This queue allows users to delete experiments that are later deemed unnecessary, specify the number of times each experiment should be repeated, and provides a button to start running the experiments in the queue.
- Experiment Data View—This view shows the result of each experiment live as it is completed in a tabular format. This view also allows the user to download the results as a Comma-Separated-Values (CSV) file. Moreover, this view also shows a progress bar to give the user an idea about how many experiments have been completed and how many more remain.

6.3.1.2 The Service Layer

This layer allows the frontend layer to communicate with the simulator layer by providing the necessary APIs. The configurations of the experiment entered through the frontend layer are fed to the simulator via this layer. This layer also communicates the results of the experiment from the simulator layer to the frontend layer. The modules contained in this layer are as follows:

- REST API Service—This provides REST API services to be consumed by the frontend layer. Users can configure experiments, start experiments and get experiment results using these REST API services. The experiment configurations sent to this service by the frontend are also persisted in a database in the data layer.
- Websocket—This allows live experiment results to be streamed to the frontend layer so that users can view the experiment results in a GUI that gets updated automatically.

- Message Queue—This is used to capture the experiment results from the simulator layer. This allows process-to-process communication between the server and the simulator. The experiment results in the message queue are also persisted in a database in the data layer.

6.3.1.3 The Simulator Layer

This is the layer where the experiments are run. This layer gets the experiment configuration from the service layer, runs the experiments, and communicates the results of the experiments back to the service layer using the message queue. This layer consists of the following modules:

- Test Grid—This is the environment in which the robots will operate on. This is a 200 × 200cm^2 grid consisting of 10 × 10cm^2 tiles of colors black and white. The ratio between the number of black and white tiles is configurable. Moreover, this grid is bounded by walls that can be detected by the robots to avoid collisions.
- e-puck Robot—This is a small robot with a footprint of 7cm^2 that is used to sense the color of the tiles and to take part in the consensus achievement task to find the color of the majority of the tiles. When blockchain is used, this robot also acts as a miner.
- ARGoS 3—This is the simulator that controls the robots. This simulator runs the robots on the test grid and finds out if consensus has been reached or not. Apart from this, the simulator also gathers evaluation metrics such as the exit probability and consensus time and communicates them to the service layer.

6.3.1.4 The Blockchain Layer

The blockchain layer consists of the blockchain, the mining nodes, the validators, and the swarm controller. The e-puck robots in the simulator layer publish their opinion to the blockchain and receive updated opinions from the smart contract running on the blockchain. The functionality of the modules in this layer is discussed below.

- Swarm Controller—This module signs the coinbase of the miners with its own private key and distributes its public key to the miners. This allows the PoI algorithm to create a dynamically permissioned blockchain.
- Miner—The e-puck robots also act as miners who mine blocks to be added to the blockchain. When the robots publish their opinions as transactions, the miners verify these transactions and add them to a new block before sealing them with their signature.

- Validator—The e-puck robots also act as validators. The validators validate the blocks mined by the miners before adding them to their blockchain. The blocks are validated by verifying the signature found in the blocks using their coinbase and the public key of the swarm controller.
- Blockchain—The smart contract that runs the decision rule algorithm is deployed in the blockchain.

6.3.1.5 The Data Layer

The data layer consists of a database that is used to persist the experiment results so that this data can be later serialized into a different format or used as it is for data analysis. Aside from this, experiment configurations sent by the frontend to the REST API service are also persisted in the database.

6.4 System Design

6.4.1 The Choice of Design Methodology

The function-oriented design was chosen as the design methodology to design both the benchmarking tool and the PoI algorithm. This decision was heavily informed by the choice of programming languages and frameworks. The React framework that was chosen to develop the frontend application, the node.js environment that was chosen to develop the server, and the Go language that was chosen to write the PoI algorithm are all primarily function-based. The rationale behind these choices will be discussed in the Implementation chapter.

Furthermore, since this research work is chiefly about the functioning of robots in a swarm and there does not exist data to be manipulated by the blockchain consensus algorithm, the functional approach was chosen ahead of the object-oriented approach. The function-oriented design methodology also assisted in the design of the blockchain consensus algorithm as it allowed the consensus problem to be decomposed into multiple small tasks.

6.4.2 The Design of the Benchmarking Tool

Figure 9 shows the data flow diagram that shows how data flows between different components of the benchmarking tool. Accordingly, it can be seen that a user first inputs the experiment configuration to the frontend app, which is then sent to the REST API service. This data is

persisted in a database while being fed into the ARGoS 3 simulator. The simulator then configures the e-puck robots using this configuration data.

The e-puck robots sense the color of the tiles in the test grid and transact their opinion about the color to the blockchain miners. The miners verify these transactions, pack them into blocks and broadcast them to the validators. The validators validate these blocks and add them to their blockchain. The blockchain smart contract runs the decision rules and updates the e-puck robots with the new opinion. The ARGoS 3 simulator reads the opinions of the robots to decide if consensus has been reached.

Once consensus is reached, the evaluation metrics of the experiment are pushed to the message queue. These metrics are persisted in the database and emitted to the frontend using a WebSocket so that the user can view the data live.

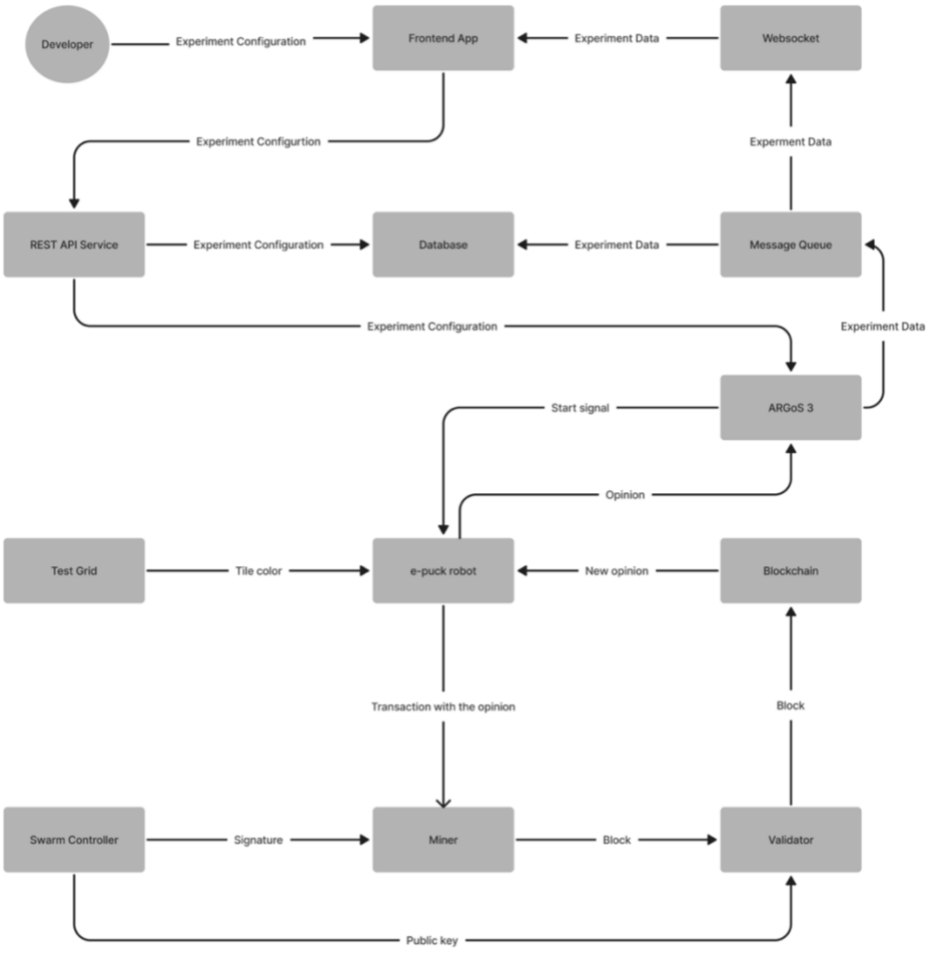


Figure 9 The data flow diagram of the benchmarking tool

6.4.3 The Design of the PoI Algorithm

The PoI algorithm creates a dynamically permissioned blockchain by enabling only authorized nodes to mine blocks. Unlike in typical Proof-of-Authority (PoA) algorithms, the authorized nodes are not declared in advance in the genesis block, nor a voting mechanism is used to allow new miners.

In order to allow new miners to be added to the blockchain network in a straightforward manner, the PoI algorithm introduces a novel swarm controller that uses a private-public key pair to sign authorized miners.

A private-public key pair is generated when a swarm controller is spun up. When a miner has to be added to the network, the miner first sends its coinbase, which is its public key that serves as its address in a blockchain, to the swarm controller. The swarm controller generates a signature using the coinbase and its private key and returns it to the miner. Then, the miner also obtains the swarm controller's public key by sending a request to the swarm controller.

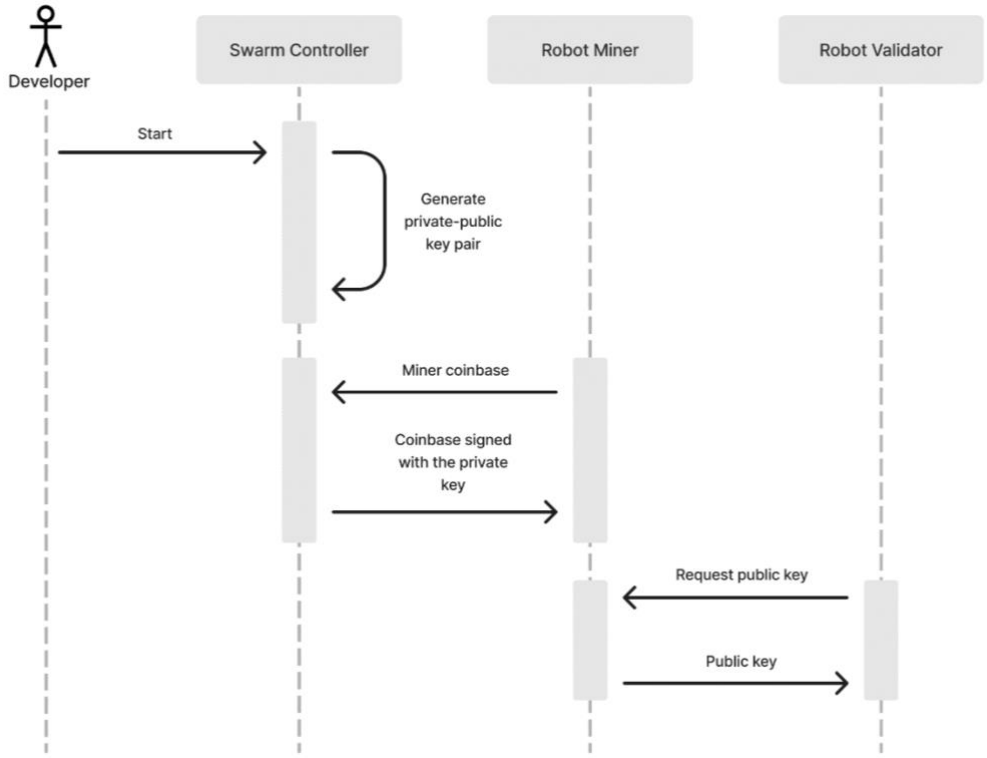


Figure 10 A sequence diagram demonstrating nodes' interaction with a swarm controller

During mining, a miner generates a timestamp and seals the block by adding its signature to the block header. The timestamp is used to decide on the priority that should be given to blocks mined by different miners. This logic shall be discussed later in this section.

When validating blocks, nodes get the coinbase and signature of the miner that mined the block and verify the signature using the swarm controller's public key. If the signature can be verified using the public key that the nodes themselves obtained from the swarm controller, then that verifies the authenticity of the miner.

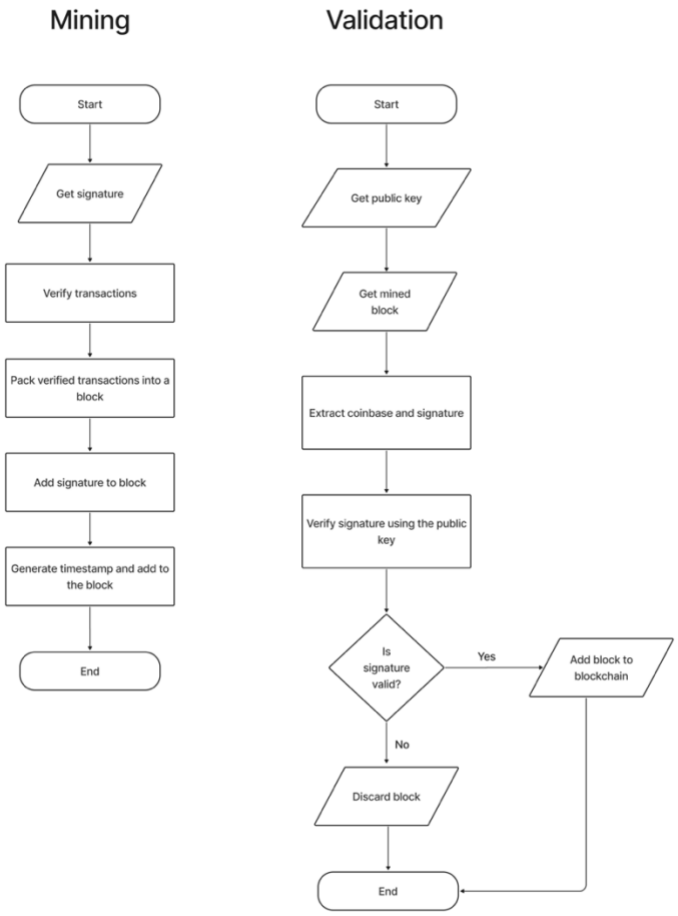


Figure 11 Flowcharts showing the mining and validation process

6.4.3.1 Signing and Signature Verification

To generate a miner's signature, the algorithm first hashes the miner's coinbase and then encrypts it with the swarm controller's private key. To verify the signature, the algorithm first hashes the coinbase of the miner of the block to be validated. Then, it decrypts the signature in the block using the swarm controller's public key and compares the decrypted value with the hash of the coinbase. If these two values match, then it can be confirmed that the signature was produced by the swarm controller. A flowchart of this process can be found in the Appendix section under **Flowchart of the Signing and Signature-Verification Process**.

6.4.3.2 Timestamp Generation

Unlike the PoW algorithm, blocks are continuously and simultaneously generated by miners in the PoI algorithm. This can result in multiple branches forming, which can affect the stability of the blockchain. To address this issue, the timestamp value is used to assign different priorities to blocks mined by different miners.

The consensus algorithm takes the possible number of robots that are likely to be found in a swarm at any given time as parameter *n*. Then, the recent *n* number of blocks in the blockchain are examined. If a block in the last *n* number of blocks in the blockchain has been mined by the current miner, then an offset value is calculated based on the position of the block on the blockchain. The closer the block is to the last block, the higher the offset value will be. This offset value is calculated for all *n* blocks and added together.

$$o = \sum_{i=1}^{n} \frac{1}{i} \times n[c_{current} = c_i]$$

Equation 6.1

Where:
n is the number of robots
$c_{current}$ is the coinbase of the current block
c_i is the coinbase of the *i*th block from the current block
o is the offset value

The higher the number of blocks and the closer they are to the last block, the higher the offset value of a miner will be. Then, random noise is added to this offset value and the final value is added to the current timestamp. If the generated timestamp is less than the timestamp of the previous block, then the offset value is added to the previous block's timestamp.

$$o_f = (\alpha \times 0) + \{(1 - \alpha)r\}$$

Equation 6.2

Where:
α is a weight
r is a random value between 1 and n.
o_f is the final offset value

When adding blocks to the blockchain, the algorithm picks the block with the earliest timestamp. Thus, the lower the offset value is, the higher the chance of getting added to the blockchain is. This gives a higher priority to miners who haven't mined many blocks recently. The random noise is added to reduce the chances of two blocks ending up with the same

timestamp value. By considering the last *n* number of blocks, each of the *n* number of robots is given a chance to mine blocks.

A flowchart describing the timestamp-generation algorithm can be found in the Appendix section under **Flowchart of the Timestamp-Generation Algorithm**.

6.4.4 Assumptions

The design makes the following assumptions.

1. Miners with a signature from the swarm controller cannot misbehave
2. Since the signature is stored in the RAM of the miner, even if a robot is captured, the signature cannot be easily extracted by the attacker.

6.5 User Interface Design

The user interface was designed to provide users with a more user-friendly experience to benchmark solutions to the collective perception scenario. To make the user interface simple and straightforward, only one screen, which is divided into three sections, is used. The interface supports both the light mode and dark mode to cater to different types of user preferences.

Figure 12 A low-fidelity wireframe of the user interface

6.6 Chapter Summary

This chapter discussed the design and architecture of the benchmarking tool and the design of the PoI algorithm. The layered architecture diagram expounded the five layers of the benchmarking tool and how they interact with one another. The data flow diagram elucidated

how data flows between different components of the benchmarking tool. The design of the PoI algorithm was explained through the sequence diagram and the flowcharts. The flowcharts also illustrated how the swarm controller produces signatures and how they are verified. Besides, the design of the timestamp generation logic was also explained using a separate flowchart. Finally, the design of the user interface was also discussed. This chapter also rationalized the design decisions that were taken while designing both the benchmarking tool and the PoI algorithm.

Chapter 7 Implementation

7.1 Chapter Overview

This chapter delves into the implementation of the benchmarking tool and the Proof-of-Identity (PoI) algorithm and discusses the languages, frameworks, libraries, and technologies used. The implementations are used to test the hypothesis of this research project. The literature review, functional and non-functional requirements gathered, and the designs all informed the implementation of the PoI algorithm and the benchmarking tool. The rationale behind the technological choices made is explained in this chapter while demonstrating how the designs were converted to code. Additionally, this chapter also discusses the salient code segments.

7.2 Technology Selection

7.2.1 Technology Stack

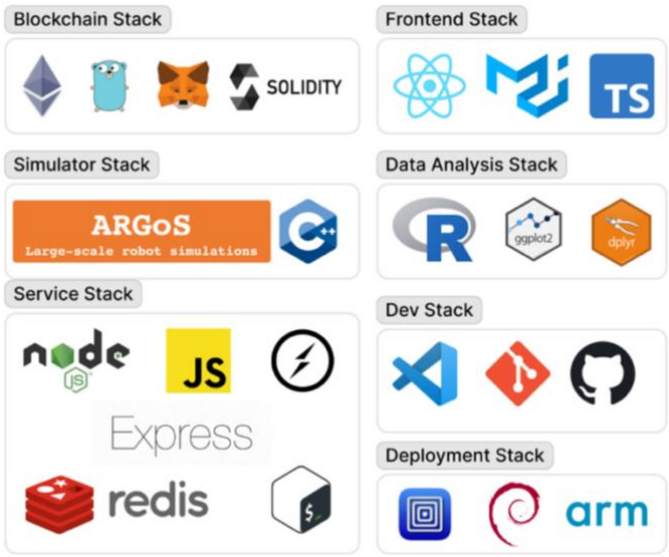


Figure 13 The technology stack that was used in the research project

Stack	Technology	Uses and Rationale
Blockchain Stack	Ethereum	Ethereum is the blockchain framework used to run the blockchain. It was chosen as it is the framework used in the research work this research work is based on.
	Go	Go is the language that was used for writing the PoI consensus algorithm. Since Ethereum is written using Go, this language was used to write the consensus algorithm.

	MetaMask	MetaMask is an Ethereum blockchain wallet that was used to test the PoI consensus algorithm.
	Solidity	Solidity is a language that is used to write Ethereum smart contracts. The smart contract that runs the decision rules was written using Solidity.
Frontend Stack	React JS	React was used to build the frontend of the benchmarking tool. This framework was used due to its status as one of the most widely used frontend frameworks, and the author's familiarity with it.
	Material UI	Material UI was used as the component library in the frontend. Material UI uses Google's Material design language and has a vibrant developer community supporting it.
	TypeScript	The frontend app was written using TypeScript. It was used since TypeScript allows type checking which helps avoid bugs in the code.
Service Stack	Node.js	The backend server was written using node.js. Node.js is one of the fastest servers for building REST API applications and it allows rapid prototyping of web applications. Moreover, node.js offers rich support for WebSocket programming.
	JavaScript	JavaScript was used to write the server application since node.js is a JavaScript programming environment.
	Socket.IO	Socket.IO is a JavaScript library that allows for building real-time web applications. This is used to push experiment results to the frontend live.
	Express	Express is a node.js framework that allows to build REST API applications and is the most popular node.js framework.
	Redis	Redis is an in-memory data structure store that can be used as a message broker. This is used to push experiment data from the ARGoS 3 simulator to the node.js server. This was chosen as it offers excellent support to both C++ and node.js.
	Bash	Bash was used to write scripts that were used to generate the ARGoS configuration file and automate the experiments.

Simulator Stack	ARGoS 3	ARGoS 3 was used to simulate the collective perception scenario. This was chosen as this was the simulator used by previous research works. This allowed code reuse and made sure the test results were comparable.
	C++	C++ was used as ARGoS 3 is based on C++. This was used to program the e-puck robots to communicate with the blockchain and to engage in consensus achievement.
Data Analysis Stack	R	R was used to analyze the experiment data. R is a popular language for data analysis and offers a rich ecosystem of libraries for data manipulation and visualization.
	ggplot2	ggplot2 is an R package that is used to produce data visualization.
	dplyr	This is an R package that allows to implement most of the commonly used R use cases easily.
Dev Stack	Visual Studio Code	Visual Studio Code was used for development as this is a powerful open-source code editor that offers support for C++, Go, TypeScript, JavaScript, and many other languages.
	Git	Git was used for version control.
	GitHub	GitHub was used to host the code repositories in the cloud.
Deployment Stack	UTM	UTM is a free, open-source virtualization software for macOS that is based on QEMU. This was used since the testing and development were carried out on a MacBook laptop and the ARGoS 3 simulator offers better support for Linux. Thus, UTM was used to run Debian virtually.
	Debian	Debian Linux distro was used to run the ARGoS 3 simulator as it supports the ARM64 architecture better.
	ARM64	The simulator was run on an ARM64 CPU virtually as the MacBook laptop that was used for testing used Apple's ARM64-based M1 chip.

Table 27 Technologies used and the rationalization

7.3 Implementation of the Benchmarking Tool

The JavaScript implementation of the backend of the benchmarking tool on node.js can be found in the Appendix section under **Code of the Backend of the Benchmarking Tool**.

7.4 Implementation of the Swarm Controller

The application code of the swarm controller written using Go is provided here.

```
var publicKey string = ""
var privateKey string = ""

func getPublicKey(c *gin.Context) {
    key, _ := getKeys()
    c.IndentedJSON(http.StatusOK, gin.H{"publicKey": key})
}

func getAccountSignature(c *gin.Context) {
    coinbase := c.Param("coinbase")
    sign := signAccount(coinbase)

    c.IndentedJSON(http.StatusOK, gin.H{"signature": sign})
}

func signAccount(coinbase string) string {
    _, key := getKeys()
    privateKey, err := decodePrivateKey(key)
    if err != nil {
        panic(err)
    }

    msgHash := sha256.New()
    msgHash.Write([]byte(coinbase))
    msgHashSum := msgHash.Sum(nil)
    signature,   := rsa.SignPKCS1v15(rand.Reader, privateKey, crypto.SHA256, msgHashSum)

    return base64.StdEncoding.EncodeToString(signature)
}

func generateKeys() *rsa.PrivateKey {
    // Generate RSA key.
    key, err := rsa.GenerateKey(rand.Reader, 4096)
    if err != nil {
        panic(err)
    }

    return key
}

func saveKeys(key *rsa.PrivateKey) {
    // Extract public component.
    publicKey = encodePublickey(&key.PublicKey)

    // Extract private component.
    privateKey = encodePrivateKey(key)
}

func encodePublickey(pubKey *rsa.PublicKey) string {
    // Marshal public key to PEM format.
    publicKeyPEM := pem.EncodeToMemory(&pem.Block{
        Type:  "RSA PUBLIC KEY",
        Bytes: x509.MarshalPKCS1PublicKey(pubKey),
    })

    return string(publicKeyPEM)
}

func encodePrivateKey(privKey *rsa.PrivateKey) string {
    // Marshal private key to PEM format.
    privateKeyPEM := pem.EncodeToMemory(&pem.Block{
        Type:  "RSA PRIVATE KEY",
        Bytes: x509.MarshalPKCS1PrivateKey(privKey),
    })

    return string(privateKeyPEM)
}

func retrieveKeys() (string, string) {
```

```
    return publicKey, privateKey
}

func decodePrivateKey(key string) (*rsa.PrivateKey, error) {
    r := strings.NewReader(key)
    pemBytes, err := ioutil.ReadAll(r)

    if err != nil {
        return nil, err
    }

    block, _ := pem.Decode(pemBytes)
    if block == nil {
        return nil, errors.New("failed to decode PEM block containing the key")
    }

    if key, err := x509.ParsePKCS1PrivateKey(block.Bytes); err == nil {
        return key, nil
    }

    return nil, err
}

func getKeys() (string, string) {
    pubKey, privKey := retrieveKeys()

    if pubKey == "" || privKey == "" {
        saveKeys(generateKeys())

        return retrieveKeys()
    }

    return pubKey, privKey
}

func main() {
    router := gin.Default()
    router.GET("/getPublicKey", getPublicKey)
    router.GET("/getSignature/:coinbase", getAccountSignature)

    router.Run(":8080")
}
```

7.5 Implementation of the PoI Algorithm

The code of the PoI algorithm written using Go is provided below. The “extra” attribute in the block header was used to add the signature of the miner to the block. The code of the PoI algorithm can be found in the Appendix section under **Code of the Proof-of-Identity Algorithm**.

The following code creates an API that allows robots to obtain their signature from the swarm controller.

```
type API struct {
    poi *PoI
}

func (api *API) GetSignature(coinbase string) {
    resp, err := http.Get(api.poi.config.SwarmControllerURI + "/getSignature/" + coinbase)
    if err != nil {
        panic(err)
    }

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        panic(err)
```

```
    }

    var sign SignatureResponse
    err = json.Unmarshal(body, &sign)

    if err != nil {
        panic(err)
    }

    signData, err := base64.StdEncoding.DecodeString(sign.Signature)

    if err != nil {
        panic(err)
    }

    api.poi.SetSignature(signData)
}
```

7.6 User Interface of the Benchmarking Tool

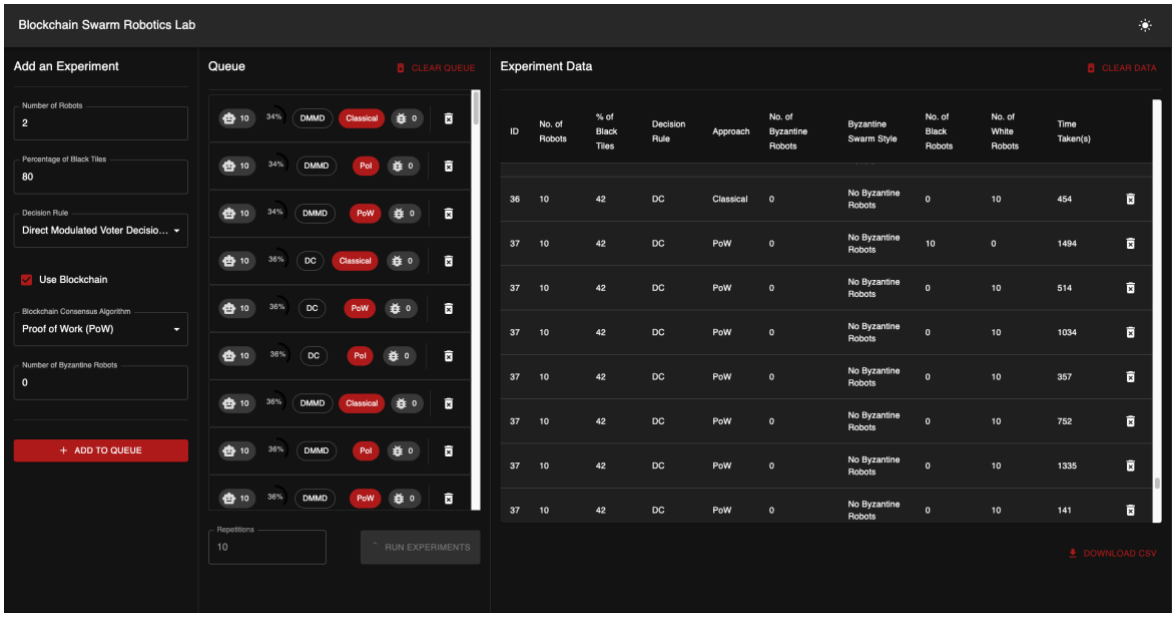


Figure 14 The user interface of the benchmarking tool

7.7 Chapter Summary

This chapter discussed the technologies that were used in the implementation of the research prototype and the rationale behind those choices. The code snippets of important functionalities of the benchmarking tool along with those of the swarm controller and the PoI algorithm were also provided in this chapter. Finally, a screen capture of the user interface of the benchmarking tool was also furnished.

Chapter 8 Testing

8.1 Chapter Overview

This chapter elaborates on how the research prototype was tested, the testing methodologies, and how the test results were used to validate the fulfillment of the functional and non-functional requirements defined in the
Software Requirements Specification chapter. First, the objectives of the tests and test criteria are discussed followed by the specification of the collective perception scenario used for testing. Then, technical details of the test environment are provided before presenting the test results. Next, this chapter discusses the findings of the tests carried out on the Proof-of-Identity (PoI) algorithm, the outcomes of the end-to-end tests, and how the completion of the functional and non-functional requirements was validated through tests. Finally, the performance test carried out on the user interface, and the limitations of the tests are discussed.

8.2 Test Objectives

The tests were carried out to test the validity of the research hypothesis and to confirm the success of the aim of this research. Besides, tests were also used to validate the fulfillment of the functional and non-functional requirements.

The main objectives of the tests carried out are detailed below:

- To benchmark the solution so that the performance can be compared against those of the other solutions.
- To verify that every functionality of the benchmarking tool works as expected.
- To verify if all the “Must have” and “Should have” functional requirements as defined by the MoSCoW technique are fulfilled by the research prototype.
- To verify if all the “Must have” and “Should have” non-functional requirements as defined by the MoSCoW technique are fulfilled by the research prototype.
- To test the performance, quality, and correctness of the benchmarking tool’s user interface.
- To ensure the solution is free of bugs and technical glitches.

8.3 Test Methods

The following test methods were used to fulfill the test objectives.

Method	Description
Benchmarking	The solution to the collective perception scenario should be tested to compare its performance with existing solutions.
Performance and security testing	The PoI algorithm should be tested to verify if it fulfills the performance and security requirements.
End-to-end testing	The benchmarking tool should be tested end-to-end manually to verify if it is functional and free of bugs.
UI performance testing	The performance of the user interface of the benchmarking tool should be tested to make sure it provides a fluid user experience.

Table 28 The test methods used

8.4 Benchmarking

8.4.1 Collective Perception Experiment

The collective perception scenario was used to benchmark the research prototype. The collective perception scenario involves a fixed number of robots coming to a consensus on the color of most tiles in a grid.

For the purposes of this research, 10 e-puck robots were deployed in a 200cm x 200cm grid bounded by four walls. The grid had 400 tiles each of area 10cm x 10cm. The tiles were either black or white in color and the ratio between the number of black and white tiles determined the difficulty of the challenge. The difficulty of the challenge is given by the following equation:

$$\rho_b = \frac{b}{w}$$

Equation 8.1

Where:

ρ_b is the difficulty of choosing white as the best opinion
b is the percentage of black tiles
w is the percentage of white tiles

At the beginning of the experiment, one half of the robots started with the opinion black, and the other half started with the opinion white. During the experiments, robots changed their opinions based on the decision rules used and the experiment ended when all robots had the

same opinion. In the experiments performed, white was always the color of most of the tiles. This was done to ensure the results of these experiments could be compared to those of the existing research works.

The experiments were executed in discreet time steps called ticks with 10 ticks forming a second. During the experiment, two robots could communicate with one another only when the distance between them was under 50cm.

The experiments had the following configurable parameters, and experiments were run for every value of each of these parameters. The parameters and the values they took are given in Table 29.

Parameter	**Values**
Difficulty	0.52, 0.56, 0.61, 0.67, 0.72, 0.79, 0.85, 0.92
Decision Rules	DMMD, DMVD, DC
Approach	Classical, Proof of Work (PoW), PoI

Table 29 The experiment parameters and their values

The following metrics were used for benchmarking:

1. Exit probability—The number of correct consensus decisions over the total number of runs.
2. Consensus time—The time taken by a swarm to achieve the correct consensus.

Tests were run for the following approaches:

1. Classical—The original approach used by Valentini et al. (2016).
2. PoW—The blockchain-based approach used by Strobel, Ferrer and Dorigo (2018) using the PoW consensus algorithm.
3. PoI—The blockchain-based approach used by Strobel, Ferrer and Dorigo (2018) using the **PoI** consensus algorithm.

Thus, altogether, 72 different types of experiments were planned. To avoid random errors and variations, each type of experiment was repeated 10 times. Consequently, 720 experiments were run in total.

8.4.2 Experiment Setup

The experiments were run on a virtual machine running on a macOS host. The details of the virtual machine and the host machine are furnished below.

	Component	Model/Type/Capacity
Virtual Machine	CPU	ARM64
	Operating System	Debian 11.3.0
	RAM	14GB
	Hypervisor	QEMU 7.0 ARM Virtual Machine
Host Machines	CPU	Apple M1 Pro
	Operating System	macOS Monterey
	RAM	16GB (Unified memory)

Table 30 Experiment setup

8.4.3 Benchmarking Results

Figure 15 shows the exit probability obtained for the three decision rules using the classical, PoW, and PoI approaches on a column graph.

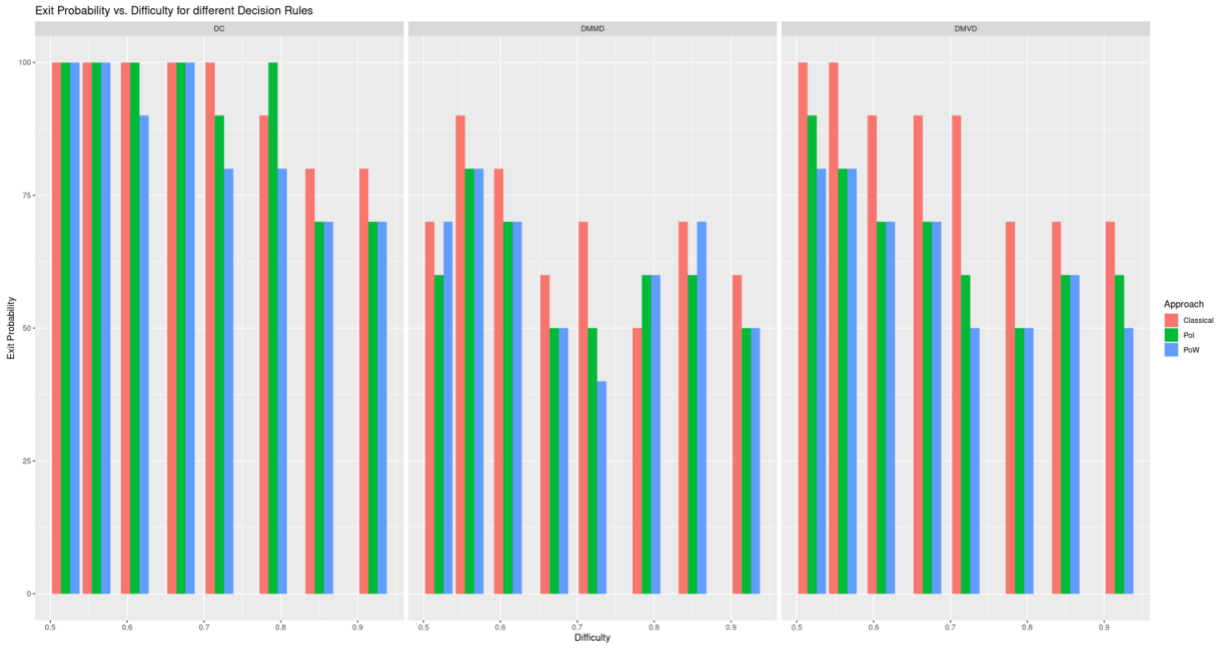


Figure 15 Exit probability for different decision rules and approaches

Figure 16 shows the consensus time obtained for the three decision rules using the three different approaches on a box plot.

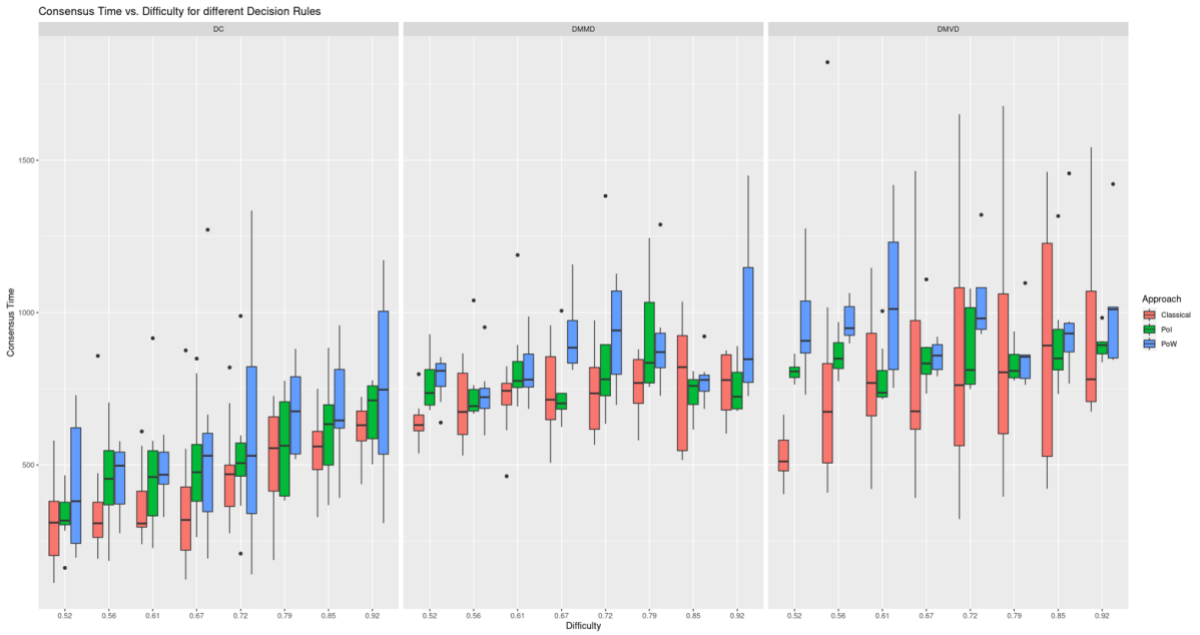


Figure 16 Consensus time for different decision rules and approaches

The results obtained for the classical as well as the PoW approaches were mostly consistent with the findings of Strobel, Ferrer and Dorigo (2018). The DC decision rule with the classical approach showed the highest exit probability under all but one difficulty settings. The DC decision rule along with the classical approach also produced the fastest consensus time.

The DMVD decision rule with the classical approach showed a steady decline in exit probability with the rise in difficulty whereas the DMMD rule, though showed an overall decline, has comparatively more variability. Overall, as far as the exit probability was concerned, the blockchain approaches performed worse than the classical approach in comparison. Both the PoI and PoW approaches showed greater parity even though the PoI performed marginally better under certain circumstances.

The consensus time of both the classical approach and the blockchain approaches steadily increased with difficulty for the DC decision rule. However, even though the classical approach showed a similar steady increase for both the DMMD and DMVD decision rules, the consensus time of the blockchain approaches was largely disaffected by the difficulty. This observation is consistent with that of Strobel, Ferrer and Dorigo (2018).

However, unlike it was the case with the exit probability, the consensus time of the PoI approach showed a significant improvement in comparison to the PoW approach for all

decision rules. Notwithstanding, the consensus time of the PoI approach was generally higher than that of the classical approach.

8.4.3.1 Discussion

The findings of the tests were very similar to the findings of Strobel, Ferrer and Dorigo (2018). Generally, the classical approach had a better exit probability than both the PoI and PoW approaches. This is due to the limitation of the blockchain approach as explained by Strobel, Ferrer and Dorigo (2018). In the classical approach, duplicate opinions from the neighbors are discarded, while in the blockchain approach, no such implementation exists. The classical approach was also faster than the PoI and PoW approaches. This is due to the delays introduced by the mining process. However, the PoI approach was shown to be faster than the PoW approach. The test results showed that the PoI algorithm, developed to nullify the Byzantine robot issue introduced by the 51%-attack threat inherent to the PoW algorithm, made consensus achievement faster while not impacting the exit probability under most circumstances and slightly improving it under some.

8.5 Testing of the PoI Algorithm

The PoI algorithm was tested against the PoW algorithm to compare the CPU usage and the blockchain consensus time. Since the e-puck robots contain powerless CPUs, the tests were run on a lightweight single-board computer to imitate the real environment.

8.5.1 Experiment

A smart contract that has a counter, which can be incremented by calling a method, was deployed on the blockchain. A node.js application using web3.js was developed and used for comparing the two consensus algorithms. The application called the smart contract method to increment the counter 100 times. The mining process was run on a single-board computer.

The consensus time was calculated by finding how long it took the counter to be incremented. In the meantime, the average CPU usage was obtained using the mpstat command by measuring the CPU usage for 5 minutes.

8.5.2 Experiment Setup

Component	Model/Type/Capacity
Device	Raspberry Pi 3 Model B
CPU	Quad Core 1.2GHz Broadcom BCM2837 64bit
RAM	1GB
Operating System	Raspbian (Linux)

Table 31 Single-board computer specifications

8.5.3 Test Results

When the PoW algorithm was used, the mining process running on the single-board computer consistently crashed due to excessive CPU usage. However, mining was successfully carried out when the PoI algorithm was used.

Consensus Algorithm	Average CPU Usage
PoW	N/A (Process crashed)
PoI	79.62%

Table 32 CPU usage

Since the PoW algorithm crashed on the single-board computer, the consensus time was measured by testing both the algorithms in the following experiment setup.

Component	Model/Type/Capacity
Device	MacBook Pro (16-inch 2021)
CPU	Apple M1 Pro
RAM	16GB (Unified memory)
Operating System	macOS Monterey

Table 33 Consensus time experiment setup

After performing 100 transactions, the average consensus time computed was as follows.

Consensus Algorithm	Average Consensus Time (ms)
PoI	3001.37
PoW	3260.7

Table 34 Average consensus time

8.5.4 Block Structure

The block structure was analyzed using the Ethereum Block Explorer tool, which demonstrated the PoI algorithm's compatibility with Ethereum's block structure.

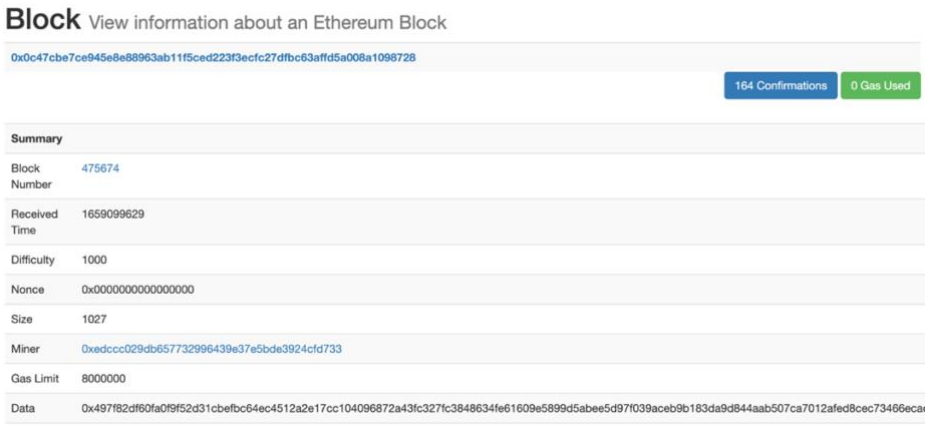


Figure 17 Block mined using PoI

8.5.5 Performance Testing

8.5.6 Security Testing

ID	Test Case	Expected Result	Actual Result	Status
ST1	A node that uses the PoW algorithm tries to join a blockchain network that uses the PoI algorithm.	Blocks mined by the node that runs the PoW algorithm should not be accepted.	Blocks mined by the node that ran the PoW algorithm were rejected by other nodes.	Passed
ST2	A node that uses the PoI algorithm but does not have access to the swarm controller tries to mine blocks.	The node should not be able to mine blocks.	The node could not be started without connecting to the swarm controller.	Passed
ST3	A node publishes blocks without its signature when PoI is used.	The blocks produced by this node should not be accepted by other nodes.	The blocks were rejected as the signature was absent.	Passed

Table 35 Security test cases and their results

8.5.7 Functional Testing

ID	Test Case	Expected Result	Actual Result	Status
FT1	The swarm administrator starts the swarm controller.	Private-public key pair gets generated and stored in memory.	The private-public key was generated and stored in memory.	Passed

FT2	A node sends a GET request to the get-signature endpoint with its coinbase.	The node gets its signature as the response.	The node received the signature and stored it in its memory.	Passed
FT3	A node sends a GET request to the get-public-key endpoint.	The node gets the public key of the swarm controller.	The node received the public key and stored it in its memory.	Passed
FT4	A miner starts mining blocks without getting the signature.	The miner is prevented from mining and an error is shown.	The miner was not allowed to start, and an error was shown.	Passed
FT5	A miner starts mining blocks after obtaining the signature.	The mined blocks have the signature and a timestamp.	The mined blocks had the signature and a timestamp.	Passed
FT6	A validator tries to validate blocks without getting the public key.	The validator is not allowed to start, and an error is shown.	The validator was not allowed to start, and an error was shown.	Passed
FT7	A validator validates blocks after getting its public key.	The validator validates blocks by verifying the block signature.	The validator validated blocks.	Passed

Table 36 Functional test cases and their results

8.6 End-to-end Testing

The end-to-end tests carried out on the benchmarking tool and the results are furnished in the Appendix section under **End-to-end Test Results**.

8.7 Validating the Fulfillment of the Requirements

The functional and non-functional requirements and the relevant test cases that were used to validate the fulfillment of these requirements are presented in the Appendix section under **Requirement Fulfillment Validation**.

8.8 Performance Testing of the User Interface

The lighthouse tool available in Microsoft Edge was used to test the performance of the user interface of the benchmarking tool. The interface scored 93 on performance.

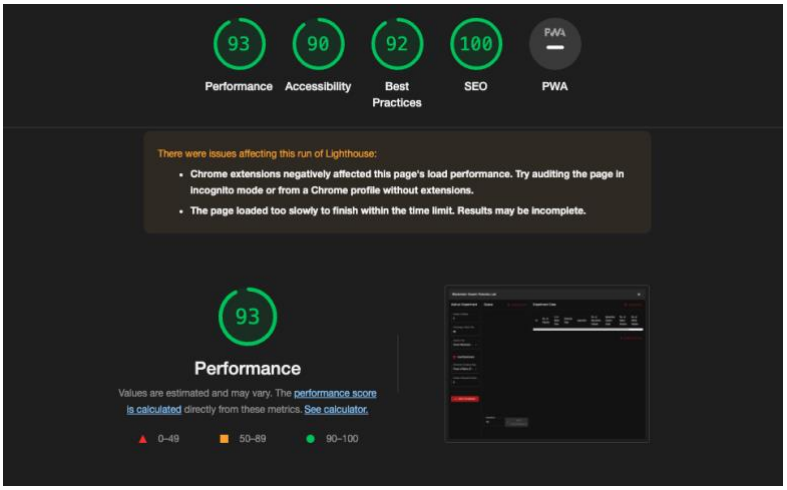


Figure 18 Lighthouse score for the user interface

8.9 Test Limitations

Due to time constraints and power cuts imposed in Sri Lanka, where the author is based in, the number of robots used in the experiments had to be reduced to 10 from 20. Besides, the experiments were repeated only 10 times even though previous research works had repeated the experiments 40 times. This was due to the smaller timeframe of this research work.

This truncation allowed the 720 experiments to be completed within the expected timeline. Although the number of robots is less likely to cause any greater discrepancies in the findings, by repeating the experiments a greater number of times, random errors and variations could have been greatly reduced.

8.10 Chapter Summary

This chapter discussed how the research prototype was tested, the testing methodologies, and how the test results were used to validate the fulfillment of the functional and non-functional requirements. The objectives of the tests and test criteria were defined followed by the specification of the collective perception scenario. Then, technical details of the test environment were provided. Subsequently, the test results were furnished, and the tests carried out on the PoI algorithm were discussed. Following this, this chapter discussed how the fulfillment of the requirements was validated through the tests carried out before examining the performance of the user interface. Finally, the test limitations were discussed.

Chapter 9 Evaluation

9.1 Chapter Overview

This chapter focuses on the evaluation that was carried out on the findings of the research work, the quality and relevance of the research work, and the value of the research work as a whole. This chapter discusses the evaluation methodology, the criteria that were used for the evaluation process, how the evaluators were chosen, and the findings of the evaluation. Besides, this chapter also delves into the completion of the functional and non-functional requirements, and the limitations of this evaluation process.

9.2 Evaluation Methodology

The evaluation was done by both the author and experts. Evaluation by the experts was carried out quantitatively and qualitatively. A short video presentation explaining the research work and its findings was produced and shared with the experts along with a feedback form. Later, a thematic analysis was performed on the feedback received from the experts.

9.3 Evaluation Criteria

The quantitative evaluation utilized the following criteria to evaluate the research work. This was done by providing the experts with multiple-choice questions to answer in the feedback form.

Criteria	**Purpose**
The novelty of the research problem	This criterion is used to evaluate if the research problem this research tries to solve is novel and original.
The novelty of the research solution	This is used to evaluate if the solution provided is novel and original.
Quality of the research work	This evaluates if the solution solves the research problem and if so, how good the solution is. Besides, this also evaluates the overall quality of the research work.

Table 37 Criteria for quantitative evaluation

Qualitative evaluation was done both by the author and experts. The questions that were posed centered around the criteria defined in Table 38. The thematic analysis, which was subsequently carried out following the interviews, factored in these criteria.

Criteria	Purpose
Research gap, problem identified, and their relevance	This evaluates if the research gap chosen is significant and relevant, and if the problem identified is valid and timely.
Literature review and its extensiveness	This evaluates the extensiveness of the literature review, its depth, and scope.
Solution design and implementation	This investigates if the solution design and implementation meet the expected quality and solve the identified problem.
Tests, their results, and findings	This evaluates if the tests carried out were adequate, and the quality of the data analysis.
Generalizability of the blockchain consensus algorithm	This evaluates if the blockchain consensus algorithm is generalizable to be used for other applications.

Table 38 Criteria for qualitative evaluation

9.4 Self-Evaluation

The author performed an evaluation of the research work based on the evaluation criteria defined.

Criteria	Evaluation
Research gap, problem identified, and their relevance	This research introduced a solution to the 51%-attack threat associated with the blockchain solution proposed by Strobel, Ferrer and Dorigo, (2018). During the literature survey, it was found that security issues are a major impediment in applying swarm robotics in realms such as defense. The survey also discovered that blockchain offered a promising solution space to address the threat posed by Byzantine robots. However, existing solutions were found to be vulnerable to the 51% attack and less performant compared to classical solutions. Thus, this research work identified the above problems as the gaps in existing research work and proposed to solve these issues. Blockchain-based solutions, in their present form, are not practically applicable to the real world even though they have been proven to be theoretically useful. By solving the research gap, blockchain solutions can be made more practically useful in the real world. This vouches for the relevance of the research gap identified.

Literature review and its extensiveness	The literature review extensively studied the background of swarm robotics, the classification of swarm robotic behaviors, consensus achievement, collective perception, and consensus achievement strategies. Further, blockchain was studied and a taxonomy was introduced to better organize the existing body of work on the applications of blockchain in swarm robotics. During this process, the various blockchain consensus algorithms available were also analyzed.
Solution design and implementation	The Proof-of-Identity (PoI) algorithm was designed as a simple algorithm light enough to run on low-powered devices while allowing the creation of a dynamically permissioned blockchain. The algorithm introduced the novel concept of a swarm controller to deploy miners and authenticate them using a private-public key pair. Moreover, an algorithm was also introduced to avoid the formation of branches by assigning different priority values to blocks.
Tests and their results and findings	The tests compared the performance of the PoI algorithm with that of the Proof-of-Work (PoW) algorithm and the classical solution for all three decision rules. Each test case was repeated 10 times to avoid random errors. In addition, manual test cases were also developed, and the benchmarking tool was tested end-to-end to ensure the proper functioning of the tool. Finally, the PoI consensus algorithm was compared with the PoW algorithm by testing them both on a low-powered single-board computer. The findings were carefully analyzed, and the superiority of the PoI algorithm was affirmed.
Generalizability of the blockchain consensus algorithm	Even though PoI was designed to solve research gaps in the swarm robotics domain, this algorithm can be easily generalized to create general-purpose dynamically permissioned blockchains. Moreover, this algorithm is also proven to be able to run on lightweight devices.

Table 39 Self-evaluation

9.5 Selection of Experts

Feedback from nine experts was solicited to better evaluate this research work. The experts were selected based on four categories. The four categories are described in Table 40.

Category ID	Category	No. of Experts
1	Blockchain experts from the industry	2
2	Swarm robotics experts from the industry and academia	4
3	Researchers in robotics	1
4	Industry professionals with an interest in swarm robotics or blockchain	2

Table 40 Categories of experts

The evaluators included one professor in swarm robotics—who is one of the pioneers in the field of swarm intelligence and a founding editor of a high-impact journal in the field, one post-doctoral researcher in blockchain-based swarm robotics, two researchers in swarm robotics, two experts in blockchain, one researcher in robotics, and two industry professionals with interests in swarm robotics and blockchain.

9.6 Evaluation Results

9.6.1 Qualitative Evaluation

A thematic analysis was conducted over the responses received from the evaluators and the results are furnished in Table 41.

Theme	Category ID	Evaluation
Research gap and problem	1	Consensus mechanisms need to be improved to keep up with the fast growth of blockchain applications. So, the research problem is very relevant.
	2	The problem domain is well described, and the research gap clearly identifies the shortcomings of the research output of Strobel, Ferrer and Dorigo (2018). As an improvement, the research work of Pacheco, Strobel and Dorigo (2020) could have been cited.
	3	The research gap has been clearly identified and the author has formulated the research questions very well.
	4	The research gap and problem are very valid and relevant.

Literature review	1	The literature review was detailed and was carried out in depth.
	2	The literature review is very detailed. The existing literature on swarm robotics has been discussed.
	3	The author has gone through the literature thoroughly and identified the problems correctly.
	4	The literature review is very good. It would have been better if the reviewed papers had been referenced in the slides.
Solution design	1	The design of the algorithm is good and appreciable.
	2	The design seems fair, and the proposed protocol has been implemented correctly. The solution is well presented as well. However, the swarm controller introduces centralization and that can be a concern. The engineering side of the project such as the design and implementation is well done although the scientific side of the project may seem incomplete. “The development and implementation of a consensus protocol is certainly very challenging, and the student managed to implement a working system. The video description is well-made, clearly structured, and easy to follow. Finally, the presented user interface is well-designed and seems to be easy to use”.
	3	The solution design is impressive, and the UI is excellent for an academic project.
	4	The design and implementation of the solution are very good. The UI could have included data visualization as well.
Testing	1	The tests seem adequate, but a practical proof of concept should be done to study the PoI and its shortcomings.
	2	The tests establish the superiority of the PoI algorithm. However, to better test the immunity to 51% attacks, what happens when valid signers with more than 51% of the “signature power” collaborate should be studied.
	3	The tests made good comparisons and expressed the usefulness of the research very well.

	4	It is good that the algorithm was tested on a low-powered device such as Raspberry Pi 3. The tests and their findings are good.
Generalizability of PoI	1	A lot of use cases can be seen for this algorithm.
	2	It is possible that PoI has applications outside swarm robotics.
	3	The PoI algorithm seems generalizable.
	4	The algorithm seems generalizable. However, centralization is introduced through the swarm controllable, which may not be desirable for certain applications.

Table 41 Qualitative evaluation

The professor who is a preeminent expert in swarm robotics noted that though they could identify limitations of the research work, they "believe the work done to be pretty good for a master student".

9.6.2 Quantitative Evaluation

The criteria and the evaluation results are presented in Table 42.

Criteria	Evaluation Results
The novelty of the research problem	
The novelty of the research solution	
Quality of the research work	 Exceptional: 62.5%. Very good: 25%. Good: 12.5%

Table 42 Quantitative evaluation

9.7 Evaluation Limitation

Since swarm robotics is a niche research field, the availability of expert researchers in swarm robotics is minimal. Moreover, owing to the limited adoption of this technology by the tech industry, not many industry experts could be found. In addition, even though blockchain has

many adopters in the industry, academic research interest in this domain seems to be lukewarm. On top of it, it was also hard to find researchers whose expertise straddled both blockchain and swarm robotics. As a result, experts in robotics and enthusiasts with years of experience in the industry were enlisted to evaluate this research work.

9.8 Evaluation of Functional Requirements

ID	Functional Requirement	Priority	Evaluation
FR1	Swarm administrators must be able to generate a public-private key pair.	M	Implemented
FR2	The swarm controller must be able to store the private key.	M	Implemented
FR3	The swarm controller must be able to encrypt the address of mining robots with the private key to produce a signature.	M	Implemented
FR4	The swarm controller must be able to send the signature to the robot.	M	Implemented
FR5	The swarm controller must be able to distribute the public key to the robots.	M	Implemented
FR6	The robots must be able to store the public key and signature.	M	Implemented
FR7	The robots must be able to attach their signature to the blocks they mine.	M	Implemented
FR8	The robots must be able to generate a timestamp for their block.	M	Implemented
FR9	The robots must be able to validate blocks using the public key.	M	Implemented
FR10	The consensus algorithm should avoid or reduce the number of branches created in the blockchain.	S	Implemented

Table 43 Function requirement evaluation

$$Completion\ percentage = \frac{10}{10} \times\ 100 = 100\%$$

9.9 Evaluation of Non-Functional Requirements

ID	Requirement	Priority	Evaluation
NFR1	Performance	M	Implemented
NFR2	Simplicity	M	Implemented
NFR3	Security	M	Implemented
NFR4	Usability	S	Implemented
NFR5	Compatibility	M	Implemented

Table 44 Non-functional requirement evaluation

$$Completion\ percentage = \frac{5}{5} \times\ 100 = 100\%$$

9.10 Chapter Summary

This chapter discussed in depth the evaluation methodology, and the quantitative and qualitative evaluation criteria. The author's self-evaluation of the research work was also furnished. Then, this chapter discussed how the expert evaluators were chosen and their evaluation was analyzed. Subsequently, the completion of the functional and non-functional requirements was evaluated. Finally, the evaluation limitations were discussed.

Chapter 10 Conclusion

10.1 Chapter Overview

This chapter dwells on the accomplishment of the research aim, learning outcomes and the challenges faced. Besides, the existing skills that were utilized and the new skills that were learned during this research are also discussed. Additionally, this research discusses the deviations from the original plan, limitations of this research, and possible future work. Finally, this chapter delves into the contribution to the body of knowledge.

10.2 Accomplishment of Research Aim

"The aim of this research is to design, develop, and evaluate a new blockchain consensus algorithm that will make the swarm more tolerant of Byzantine robots by resolving the 51%-attack vulnerability associated with the Proof-of-Work algorithm, and allow legitimate robots to authenticate themselves with the swarm as miners without affecting the exit probability and consensus time associated with the Proof-of-Work algorithm."

The aim of this research was successfully accomplished by designing, developing, and testing the Proof-of-Identity (PoI) algorithm that improved Byzantine fault tolerance of a swarm of robots by resolving the 51%-attack vulnerability that stems from the Proof-of-Work (PoW) algorithm while allowing legitimate robots to be added to the swarm as miners without affecting the exit probability and consensus time associated with the PoW algorithm.

10.3 Utilization of Skills Gained from the Course

Module	**Description**
Data Mining and Machine Learning	The R programming skill and data analysis skill acquired through this module helped the author analyze the experiment data and produce data visualization.
Concurrent and Distributed Systems	The knowledge gained about distributed systems was helpful in understanding blockchain better. Furthermore, the understanding of concurrent programming allowed the author to learn Go routines easily.
Advanced Software Engineering	The knowledge about design patterns and UML diagrams gained through this module helped design and architecture the solution to the research problem.

Enterprise Application Development	Knowledge about UML diagrams such as the sequence diagram and use-case diagram was used to come up with these diagrams for the solution proposed by this research.

Table 45 Utilization of skills gained from the course

10.4 Existing Skills Utilized

The following skills the author had were used during this research

- Knowledge gained through a research project on blockchain the author had worked on in the past helped the author understand blockchain and consensus algorithms better.
- The author's experience in frontend development was useful in developing the frontend application of the benchmarking tool.
- The author's knowledge about WebSockets was useful in live streaming data to the frontend.
- C++ programming skills the author had was helpful in programming the ARGoS 3 simulator.

10.5 New Skills Gained

The author gained the following skills during this research project.

- **Blockchain consensus algorithm development**—The author learned about consensus algorithms in detail and developed a new consensus algorithm.
- **Go development**—The author had to learn the Go language to develop the consensus algorithm and gained appreciable development skills in Go.
- **ARGoS 3 Simulator**—The author gained a very good understanding of the ARGoS 3 simulator and about running experiments using it.
- **Redis development**—This research helped the author learn about the Redis database and its use as a message queue.

10.6 Accomplishment of Learning Outcomes

Accomplishment	Learning Outcomes
The author learned to study and critically analyze the existing body of work in a domain as evident from the Literature Review chapter.	LO1
This research taught the author to research various technologies available, identify their pros and cons, understand the resource constraints and select the right technology. The author also learned to gather requirements to build a better solution. (Methodology and Software Requirements Specification chapters)	LO5, LO3
The author learned to design a solution using UML diagrams, identify shortcomings and iterate on the designs to produce the best possible design. (Software Requirements Specification and System Architecture and Design chapters)	LO3, LO4
The author learned to implement a design by coding the solution. The author also learned to develop test cases and evaluation strategies to test the solution. (Implementation chapter)	LO3, LO2, LO5
The author learned to evaluate the solution and benchmark them to study the effectiveness of a solution. (Testing and Evaluation chapters)	LO3, LO5
The author learned to document the design, implementation, and test results of the research work. The author also produced research papers from their research work. (Publications section)	LO5, LO2, LO6, LO4

Table 46 Accomplishment of learning outcomes

10.7 Problems and Challenges Faced

Challenges/Problems Faced	Solution
Limited documentation and community support for blockchain consensus algorithm development using Go Ethereum.	The author studied the codebase of Go Ethereum to understand the consensus algorithm. The author also developed test cases to develop the consensus algorithm through trial and error.
Longer testing time and power cuts.	The country the author lives in went through a forex crisis that led to power cuts. As a result, the tests could not be run continuously. When the tests were on while the laptop was not plugged in, significant variations

	could be observed as the CPU was throttled by the operating system to save battery power. Consequently, the number of robots used in the experiments was reduced from 20 to 10. This allowed the tests to be completed faster.
ARGoS 3 simulator not supporting macOS	Even though ARGoS 3's site claimed that the simulator runs on macOS, the author could not get the simulator to run on this operating system. So, a virtual machine running Debian was set up to use the simulator.
Vast learning curve	The author had no prior experience in blockchain consensus algorithm development and the domain of swarm robotics was new to the author. More time was invested in learning these two to gain enough knowledge to complete this project.

Table 47 Problems and challenges faced

10.8 Deviations

There were not any deviations from the original design. However, the evaluation strategy had to be modified to reduce the number of robots used in the experiments to allow the testing to accommodate hours-long power cuts experienced daily.

10.9 Limitations of the Research

1. The solution expects developers to physically protect the swarm controller from rogue robots accessing it.
2. The effectiveness of the timestamp-generation algorithm was not extensively tested.
3. The solution was tested using a simulator due to resource constraints.
4. Consensus time when the PoI algorithm was used was significantly higher than when classical solutions were used.

10.10 Future Work

- Developing an authentication mechanism to authenticate robots with the swarm controller.
- Developing blockchain-native decision rules to improve the exit probability.

- Introducing a mechanism to allow robots to take turns to mine blocks. This will greatly reduce the power expended on mining blocks that would be rejected.
- Using blockchain concepts to address the Byzantine-robot problem without using blockchain as a whole.
- Studying the effectiveness of the timestamp-generation algorithm extensively.
- Adding an authorization mechanism on top of consensus algorithms such as Clique and Stellar Consensus Algorithm and comparing their performance to Proof of Identity.

10.11 Research Contribution

10.11.1 Contribution to Technology

This research invented a new blockchain consensus algorithm that allows the creation of dynamically permissioned blockchains. This allows authorized miners to be added to a blockchain network during runtime. This algorithm has applications even outside swarm robotics.

10.11.2 Contribution to Domain

This research work improved Byzantine fault tolerance in swarm robotics by addressing the 51%-attack issue found in the existing blockchain solution without compromising on the performance. Moreover, the developed solution was also shown to perform better than the existing blockchain solution improving the practical usability of blockchain-based solutions.

Besides, this research also created a web application to benchmark solutions to the collective perception scenario. This application will help future researchers benchmark their solutions in a lot more user-friendly manner.

10.12 Chapter Summary

This chapter concludes the dissertation by discussing the accomplishment of the research aim, learning outcomes and the challenges faced. This chapter also explored the existing skills that were utilized and the new skills that were gained. Apart from this, deviations from the original plan, limitations of the research work, future work, and contribution to the body of knowledge were also discussed.

10.13 Concluding Remarks

This research contributed significantly to both the problem domain and the technical domain. The PoI algorithm is expected to have applications even outside swarm robotics. The affirmative findings would further encourage future researchers to explore the blockchain space in swarm robotics more and more. In addition, the benchmarking web application is expected to allow developers and researchers to benchmark their solutions to the collective perception scenario more easily.

This research was planned well in advance and the plans were executed immaculately throughout the project. The overall feedback received from the evaluators was positive and the scholars in swarm robotics were able to identify gaps in this research work that could lead to further research.

References

Anita, N. and Vijayalakshmi, M. (2019) "Blockchain Security Attack: A Brief Survey," in *2019 10th International Conference on Computing, Communication and Networking Technologies, ICCCNT 2019*. Institute of Electrical and Electronics Engineers Inc. Available at: https://doi.org/10.1109/ICCCNT45670.2019.8944615.

Beni, G. (2005) "From swarm intelligence to swarm robotics," *Lecture Notes in Computer Science*, 3342, pp. 1–9. Available at: https://doi.org/10.1007/978-3-540-30552-1_1.

Brambilla, M. *et al.* (2013) "Swarm robotics: A review from the swarm engineering perspective," *Swarm Intelligence*, 7(1), pp. 1–41. Available at: https://doi.org/10.1007/s11721-012-0075-2.

Brambilla, M. *et al.* (2014) "Property-driven design for robot swarms: A design method based on prescriptive modeling and model checking," *ACM Transactions on Autonomous and Adaptive Systems*, 9(4). Available at: https://doi.org/10.1145/2700318.

Camazine, Deneubourg, Franks, et al. : (2001) *Self-Organization in Biological Systems*, *Princeton University Press*. Available at: http://books.google.com/books?hl=en&lr=&id=zMgyNN6Ufj0C&oi=fnd&pg=PR7&dq=Self-Organization+in+Biological+Systems&ots=MJHsi_DFdj&sig=_JbErT5TNKlfxETFH4CI9cGx0As (Accessed: July 15, 2021).

Chen, L. *et al.* (2017) "On security analysis of proof-of-elapsed-time (PoET)," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10616 LNCS, pp. 282–297. Available at: https://doi.org/10.1007/978-3-319-69084-1_19/COVER/.

Christensen, A.L., O'Grady, R. and Dorigo, M. (2009) "From fireflies to fault-tolerant swarms of robots," *IEEE Transactions on Evolutionary Computation*, 13(4), pp. 754–766. Available at: https://doi.org/10.1109/TEVC.2009.2017516.

Couzin, I.D. *et al.* (2005) "Effective leadership and decision-making in animal groups on the move," *Nature*, 433(7025), pp. 513–516. Available at: https://doi.org/10.1038/nature03236.

Crosby, M. (2016) “BlockChain Technology: Beyond Bitcoin,” *Applied Innovation Review Issue* [Preprint], (2). Available at: http://scet.berkeley.edu/wp-content/uploads/AIR-2016-Blockchain.pdf.

Ferdous, M.S. *et al.* (2020) “Blockchain Consensus Algorithms: A Survey.” Available at: http://arxiv.org/abs/2001.07091 (Accessed: August 30, 2021).

Frisch, K. von (1993) *The Dance Language and Orientation of Bees*, *The Dance Language and Orientation of Bees*. Harvard University Press. Available at: https://doi.org/10.4159/harvard.9780674418776.

Higgins, F., Tomlinson, A. and Martin, K.M. (2009) “Survey on security challenges for swarm robotics,” *Proceedings of the 5th International Conference on Autonomic and Autonomous Systems, ICAS 2009*, pp. 307–312. Available at: https://doi.org/10.1109/ICAS.2009.62.

Karthik, S. *et al.* (2020) “Bee-Bots: A Blockchain Based Decentralised Swarm Robotic System,” *2020 6th International Conference on Control, Automation and Robotics, ICCAR 2020*, pp. 145–150. Available at: https://doi.org/10.1109/ICCAR49639.2020.9108053.

Kim, M., Kwon, Y. and Kim, Y. (2019) “Is stellar as secure as you think?,” *Proceedings - 4th IEEE European Symposium on Security and Privacy Workshops, EUROS and PW 2019*, pp. 377–385. Available at: https://doi.org/10.1109/EUROSPW.2019.00048.

Krishnamohan, T. *et al.* (2020) “BlockFlow: A decentralized SDN controller using blockchain,” *International Journal of Scientific and Research Publications (IJSRP)*, 10(3), p. p9991. Available at: https://doi.org/10.29322/ijsrp.10.03.2020.p9991.

Lamport, L., Shostak, R. and Pease, M. (1982) “The Byzantine Generals Problem,” *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3), pp. 382–401. Available at: https://doi.org/10.1145/357172.357176.

Liskov, M.C. and B. (2010) “Practical Byzantine Fault Tolerance,” in *Proceedings ofthe Third Symposium on Operating Systems Design and Implementation, New Orleans, USA*, pp. 359–368.

Mahmood, W. and Wahab, A. (2020) “Survey of Consensus Protocols,” *SSRN Electronic Journal* [Preprint]. Available at: https://doi.org/10.2139/ssrn.3556482.

Mazi`eres, D. and Mazi`eres, M. (no date) “The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus.”

Mondada, F. *et al.* (2009) “The e-puck, a Robot Designed for Education in Engineering,” *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, 1(1), pp. 59–65. Available at: https://infoscience.epfl.ch/record/135236 (Accessed: July 17, 2021).

Nakamoto, S. (2009) “Bitcoin: A Peer-to-Peer Electronic Cash System.” Available at: www.bitcoin.org.

Nguyen, T.T., Hatua, A. and Sung, A.H. (2020) *Blockchain approach to solve collective decision making problems for swarm robotics*, *Advances in Intelligent Systems and Computing*. Springer International Publishing. Available at: https://doi.org/10.1007/978-3-030-23813-1_15.

Pacheco, A., Strobel, V. and Dorigo, M. (2020) “A Blockchain-Controlled Physical Robot Swarm Communicating via an Ad-Hoc Network,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12421 LNCS, pp. 3–15. Available at: https://doi.org/10.1007/978-3-030-60376-2_1.

Pinciroli, C. *et al.* (2012) “ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems,” *Swarm Intelligence 2012 6:4*, 6(4), pp. 271–295. Available at: https://doi.org/10.1007/S11721-012-0072-5.

Saad, M. *et al.* (2020) “Exploring the Attack Surface of Blockchain: A Comprehensive Survey,” *IEEE Communications Surveys and Tutorials*, 22(3), pp. 1977–2008. Available at: https://doi.org/10.1109/COMST.2020.2975999.

Şahin, E. (2005) “Swarm robotics: From sources of inspiration to domains of application,” *Lecture Notes in Computer Science*, 3342, pp. 10–20. Available at: https://doi.org/10.1007/978-3-540-30552-1_2.

Scheidler, A. *et al.* (2016) “The k-unanimity rule for self-organized decision-making in swarms of robots,” *IEEE Transactions on Cybernetics*, 46(5), pp. 1175–1188. Available at: https://doi.org/10.1109/TCYB.2015.2429118.

Singh, P.K. *et al.* (2020) “An efficient blockchain-based approach for cooperative decision making in swarm robotics,” *Internet Technology Letters*, 3(1), p. e140. Available at: https://doi.org/10.1002/itl2.140.

Strobel, V., Castelló Ferrer, E. and Dorigo, M. (2020) “Blockchain Technology Secures Robot Swarms: A Comparison of Consensus Protocols and Their Resilience to Byzantine Robots,” *Frontiers in Robotics and AI*, 7, p. 54. Available at: https://doi.org/10.3389/frobt.2020.00054.

Strobel, V., Ferrer, E.C. and Dorigo, M. (2018) “Managing Byzantine Robots via Blockchain Technology in a Swarm Robotics Collective Decision Making Scenario,” in *International Conference on Autonomous Agents and Multiagent Systems*. Available at: www.ifaamas.org (Accessed: June 26, 2021).

Tran, J.A. *et al.* (2019) “SwarmDAG: A Partition Tolerant Distributed Ledger Protocol for Swarm Robotics,” *Ledger* [Preprint]. Available at: https://doi.org/10.5195/ledger.2019.174.

Valentini, G., Ferrante, E., *et al.* (2016) “Collective decision with 100 Kilobots: speed versus accuracy in binary discrimination problems,” *Autonomous Agents and Multi-Agent Systems*, 30(3), pp. 553–580. Available at: https://doi.org/10.1007/s10458-015-9323-3.

Valentini, G., Brambilla, D., *et al.* (2016) “Collective Perception of Environmental Features in a Robot Swarm,” in *Swarm Intelligence, 10th International Conference, ANTS 2016 Brussels, Belgium, September 7–9, 2016 Proceedings*. Springer International Publishing Switzerland 2016, pp. 65–76. Available at: https://doi.org/10.1007/978-3-319-44427-7_2.

Valentini, G., Ferrante, E. and Dorigo, M. (2017) “The best-of-n problem in robot swarms: Formalization, state of the art, and novel perspectives,” *Frontiers Robotics AI*. Frontiers Media S.A., p. 1. Available at: https://doi.org/10.3389/frobt.2017.00009.

Valentini, G., Hamann, H. and Dorigo, M. (2014) “Self-organized collective decision making: The weighted voter model,” *13th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2014*, 1(January), pp. 45–52.

Valentini, G., Hamann, H. and Dorigo, M. (2015) *Efficient Decision-Making in a Self-Organizing Robot Swarm: On the Speed Versus Accuracy Trade-Off*. DMMD; swarm robotics. Available at: www.ifaamas.org (Accessed: June 28, 2021).

Winfield, A.F.T., Harper, C.J. and Nembrini, J. (2005) “Towards dependable swarms and a new discipline of swarm engineering,” *Lecture Notes in Computer Science*, 3342, pp. 126–142. Available at: https://doi.org/10.1007/978-3-540-30552-1_11.

Zakiev, A., Tsoy, T. and Magid, E. (2018) “Swarm Robotics: Remarks on Terminology and Classification,” in *Interactive Collaborative Robotics*, pp. 291–298. Available at: https://doi.org/10.1007/978-3-319-99582-3_20.

Appendix

1. Concept Map

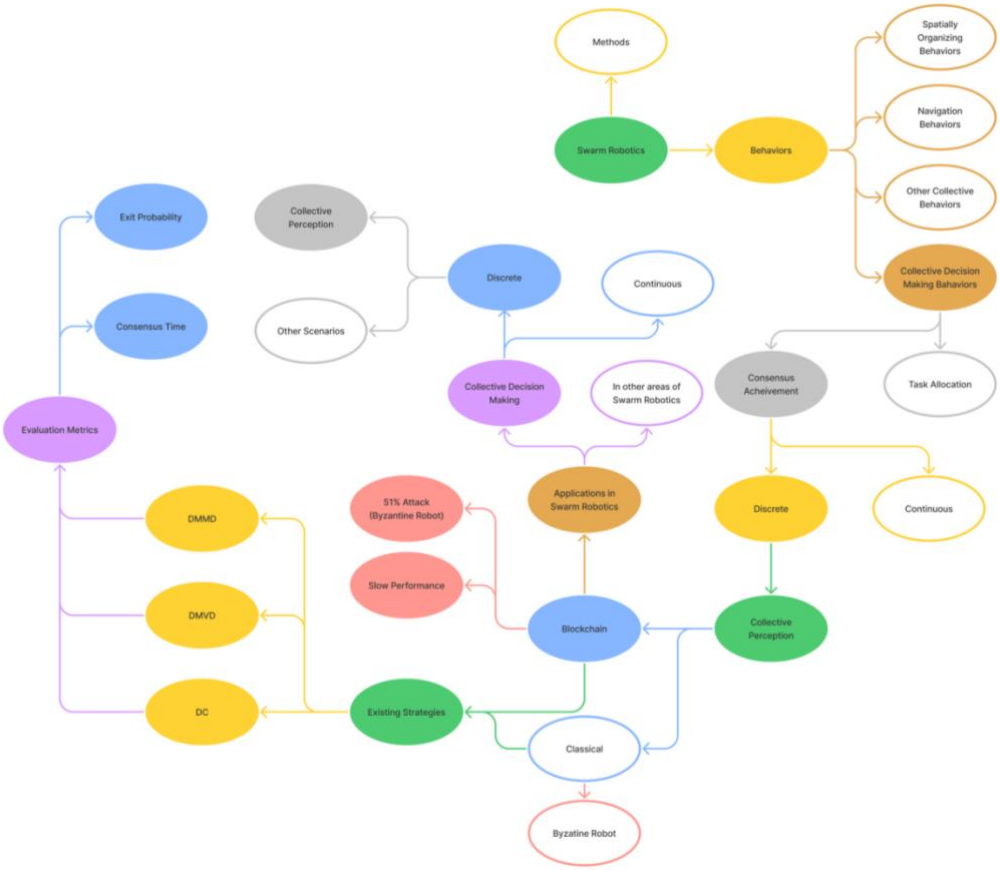
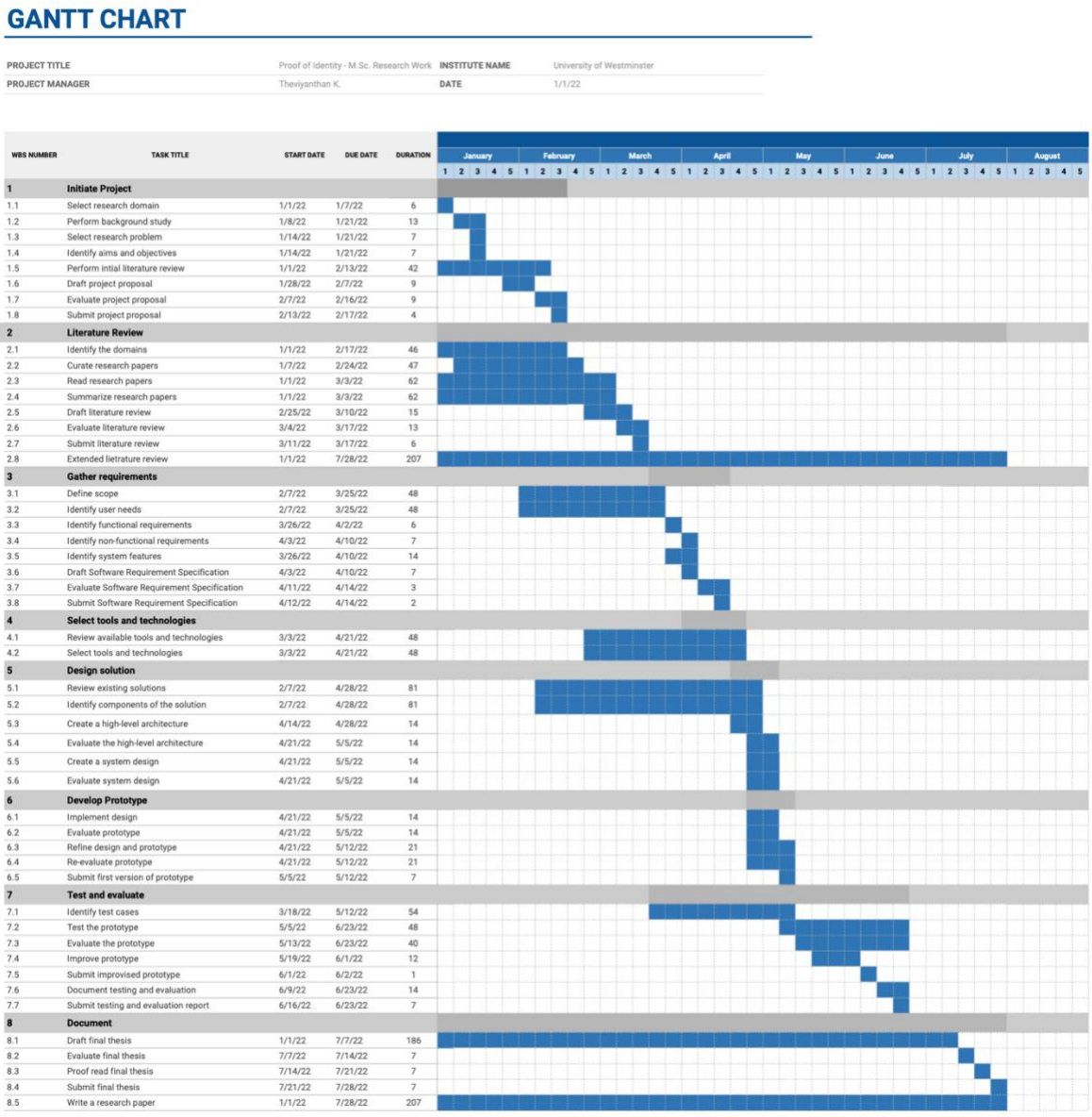


Figure 19 Concept Map

2. Gantt Chart



GANTT CHART

PROJECT TITLE	Proof of Identity - M.Sc. Research Work	INSTITUTE NAME	University of Westminster
PROJECT MANAGER	Theviyanthan K.	DATE	1/1/22

WBS NUMBER	TASK TITLE	START DATE	DUE DATE	DURATION
1	**Initiate Project**			
1.1	Select research domain	1/1/22	1/7/22	6
1.2	Perform background study	1/8/22	1/21/22	13
1.3	Select research problem	1/14/22	1/21/22	7
1.4	Identify aims and objectives	1/14/22	1/21/22	7
1.5	Perform intial literature review	1/1/22	2/13/22	42
1.6	Draft project proposal	1/28/22	2/7/22	9
1.7	Evaluate project proposal	2/7/22	2/16/22	9
1.8	Submit project proposal	2/13/22	2/17/22	4
2	**Literature Review**			
2.1	Identify the domains	1/1/22	2/17/22	46
2.2	Curate research papers	1/7/22	2/24/22	47
2.3	Read research papers	1/1/22	3/3/22	62
2.4	Summarize research papers	1/1/22	3/3/22	62
2.5	Draft literature review	2/25/22	3/10/22	15
2.6	Evaluate literature review	3/4/22	3/17/22	13
2.7	Submit literature review	3/11/22	3/17/22	6
2.8	Extended lietrature review	1/1/22	7/28/22	207
3	**Gather requirements**			
3.1	Define scope	2/7/22	3/25/22	48
3.2	Identify user needs	2/7/22	3/25/22	48
3.3	Identify functional requirements	3/26/22	4/2/22	6
3.4	Identify non-functional requirements	4/3/22	4/10/22	7
3.5	Identify system features	3/26/22	4/10/22	14
3.6	Draft Software Requirement Specification	4/3/22	4/10/22	7
3.7	Evaluate Software Requirement Specification	4/11/22	4/14/22	3
3.8	Submit Software Requirement Specification	4/12/22	4/14/22	2
4	**Select tools and technologies**			
4.1	Review available tools and technologies	3/3/22	4/21/22	48
4.2	Select tools and technologies	3/3/22	4/21/22	48
5	**Design solution**			
5.1	Review existing solutions	2/7/22	4/28/22	81
5.2	Identify components of the solution	2/7/22	4/28/22	81
5.3	Create a high-level architecture	4/14/22	4/28/22	14
5.4	Evaluate the high-level architecture	4/21/22	5/5/22	14
5.5	Create a system design	4/21/22	5/5/22	14
5.6	Evaluate system design	4/21/22	5/5/22	14
6	**Develop Prototype**			
6.1	Implement design	4/21/22	5/5/22	14
6.2	Evaluate prototype	4/21/22	5/5/22	14
6.3	Refine design and prototype	4/21/22	5/12/22	21
6.4	Re-evaluate prototype	4/21/22	5/12/22	21
6.5	Submit first version of prototype	5/5/22	5/12/22	7
7	**Test and evaluate**			
7.1	Identify test cases	3/18/22	5/12/22	54
7.2	Test the prototype	5/5/22	6/23/22	48
7.3	Evaluate the prototype	5/13/22	6/23/22	40
7.4	Improve prototype	5/19/22	6/1/22	12
7.5	Submit improvised prototype	6/1/22	6/2/22	1
7.6	Document testing and evaluation	6/9/22	6/23/22	14
7.7	Submit testing and evaluation report	6/16/22	6/23/22	7
8	**Document**			
8.1	Draft final thesis	1/1/22	7/7/22	186
8.2	Evaluate final thesis	7/7/22	7/14/22	7
8.3	Proof read final thesis	7/14/22	7/21/22	7
8.4	Submit final thesis	7/21/22	7/28/22	7
8.5	Write a research paper	1/1/22	7/28/22	207

Figure 20 Gantt chart

3. Resource Requirements

3.1. Software Requirements

Requirement	Justification
ARGoS Simulator	The consensus strategy was evaluated in a simulated environment owing to cost and time constraints. The research work carried out by Valentini, Brambilla, et al. (2016) in both the physical environment and a simulated environment demonstrated the advantages simulation has such as the ability to scale up the number of robots in a swarm easily, repeat a test

	many number of times, and the absence of external noise that affect the outcome of the experiments. The state-of-the-art ARGoS simulator will be used to simulate the swarm robotics environment as this is the simulator that has been unanimously adopted by the swarm robotics community. It is also an open-source simulator that is available free of charge and is the simulator of choice among academic researchers.
E-puck robot plugin for ARGoS	The e-puck robots will be used as members of the swarm owing to their small footprint, simple design, and the presence of the components necessary for the collective perception scenario. In addition, both Valentini, Brambilla, et al. (2016) and Strobel, Ferrer and Dorigo (2018) used the ARGoS simulator and the e-puck robots, so comparing the performance of the proposed solution will be a lot easier.
Ethereum framework	Both Hyperledger and Ethereum were considered to develop the blockchain solution. However, Ethereum was chosen as the blockchain framework to develop this solution since previous research works such as those by Strobel, Ferrer and Dorigo (2018), Nguyen, Hatua and Sung (2020) and Singh et al. (2020) used Ethereum. Thus, Ethereum is a proven framework in swarm robotics and the use of Ethereum to develop the proposed solution will allow for effective comparisons to be drawn with previous research works.
Microsoft Word	This is used for producing documents and reports. It is far more advanced than any other word processor available. Since the institute has an Office 365 subscription for students, the cost is a non-issue.
Visual Studio Code	It is a code editor that is available free of charge and offers support for many languages.
Mendeley	For managing research papers.
Linux/Mac OS	ARGoS supports only Mac and Linux.

Table 48 Software requirements

3.2. Hardware Requirements

Requirement	Justification
Core i7 processor or its equivalent	For running a simulator and a blockchain network.
16 GB RAM	For the smooth running of a development environment.

Table 49 Hardware requirements

3.3. Skill Requirements

Requirement	Justification
Ethereum blockchain development	Since the Ethereum blockchain platform will be used to develop the new consensus algorithm, Ethereum development skills will be required.
ARGoS development	Consensus achievement strategies must be programmed using the ARGoS simulator. So, this skill is necessary.

Table 50 Skill requirements

4. Flowchart of the Signing and Signature-Verification Process

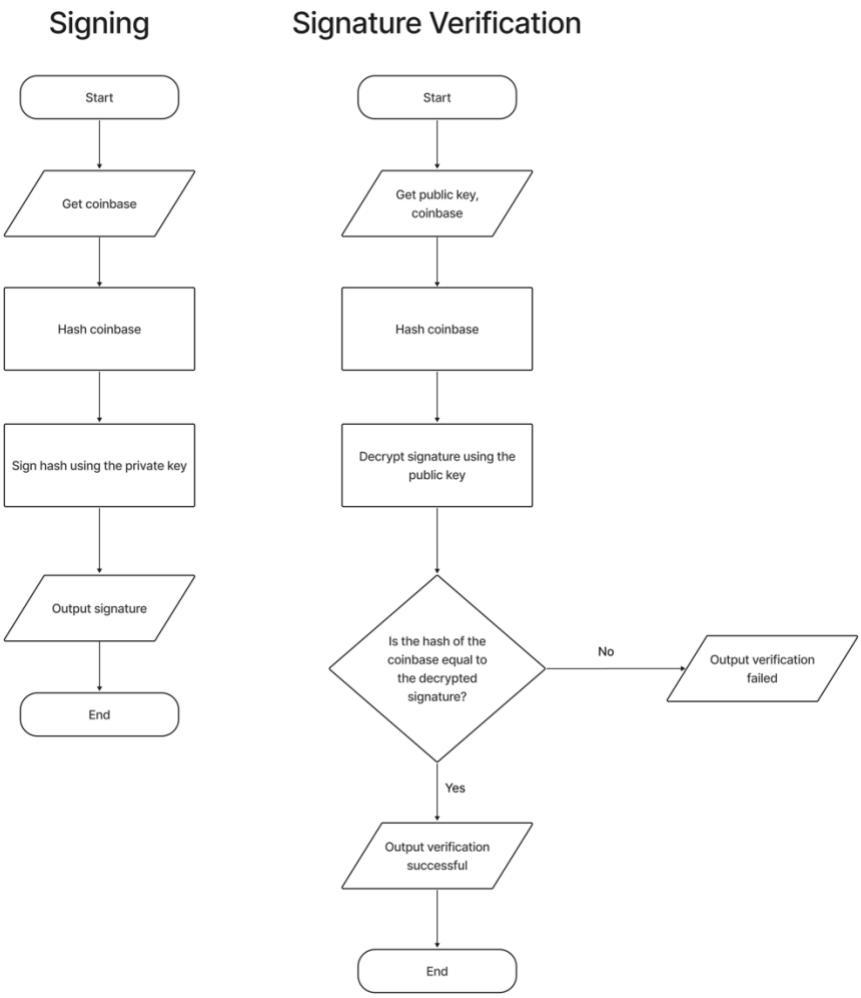


Figure 21 Flowcharts showing the signing and signature verification processes

5. Flowchart of the Timestamp-Generation Algorithm

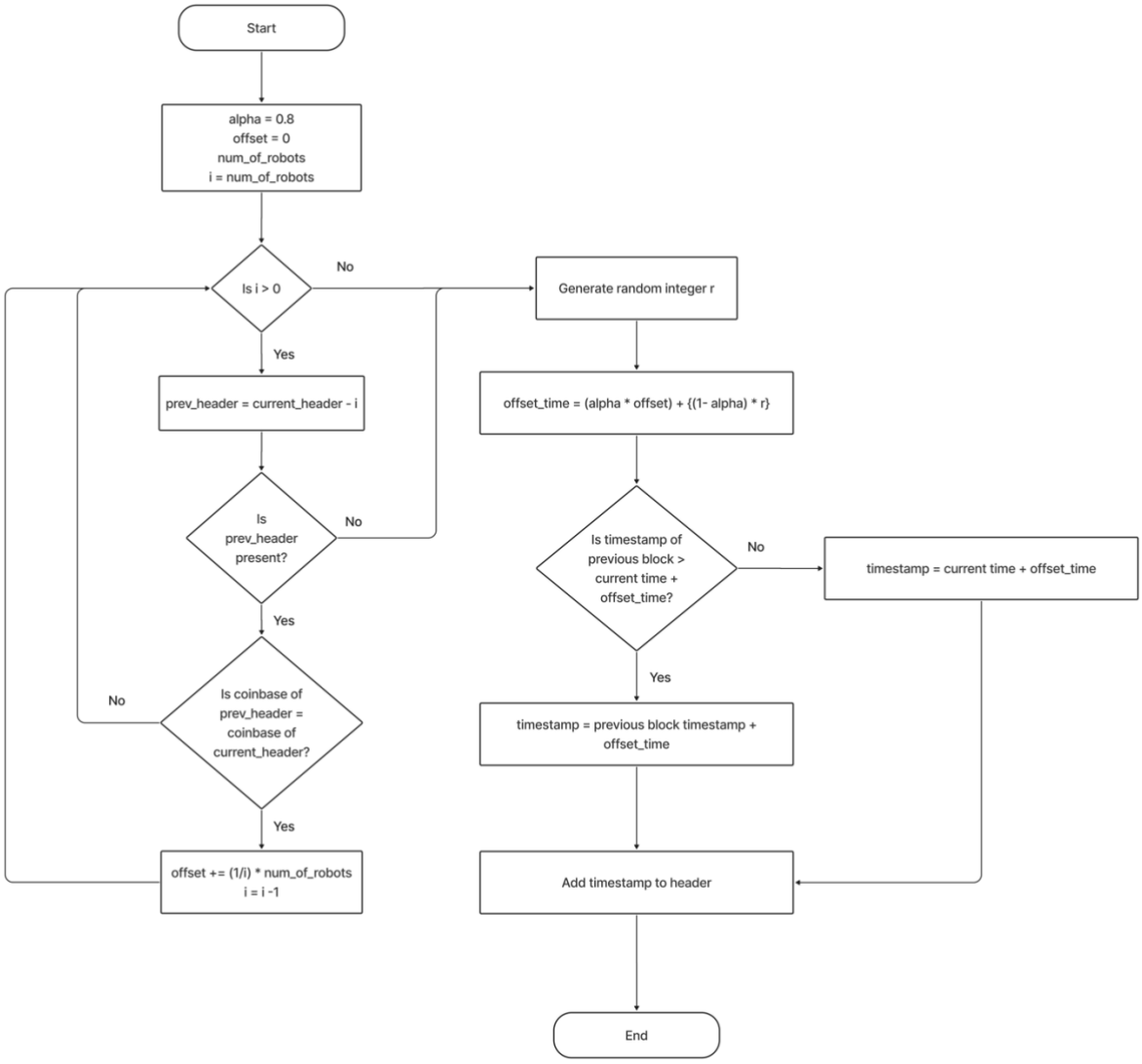


Figure 22 A flowchart explaining how a timestamp is generated

6. Code of the Backend of the Benchmarking Tool

```
const { createClient } = require("redis");
const express = require("express");
const app = express();
const http = require("http");
const server = http.createServer(app);
const client = createClient();
const { Server } = require("socket.io");
const io = new Server(server, {
    cors: {
        origin: "*"
    }
});
const { spawn } = require('child_process');
const cors = require("cors");
const JSONdb = require("simple-json-db");
const path = require("path");
var bodyParser = require('body-parser');
const { Parser } = require("json2csv");
const opts = {
    fields: [
```

```
        "secondsTaken",
        "numberOfWhites",
        "numberOfBlacks",
        "numberOfRobots",
        "isClassical",
        "consensusAlgorithm",
        "percentageOfBlackTiles",
        "decisionRule",
        "byzantineSwarmStyle",
        "numberOfByzantineRobots",
        "id"
    ]
}

const db = new JSONdb(path.join(  dirname, "..", "storage.json"));
const inputID = process.argv[ 2 ] ? process.argv[ 2 ] : 0;
console.log(inputID)
const EXPERIMENT_DATA = "experimentData";
const QUEUE = "queue";

app.use(cors());
app.use(bodyParser.json());

let lock = false;
let sockets = [];
let id = inputID;
client.on("error", (err) => console.log("Redis Client Error", err));

client.connect().then(() => {
    client.subscribe("experimentData", (data) => {
        try {
            console.log(data);
            currentExperimentData = { ...JSON.parse(data), id };
            if (db.has(EXPERIMENT_DATA)) {
                const existingData = db.get(EXPERIMENT DATA);
                existingData.push(currentExperimentData);
                db.set(EXPERIMENT DATA, existingData);
            } else {
                db.set(EXPERIMENT_DATA, [ currentExperimentData ]);
            }
            sockets.forEach(socket => {
                socket.emit("EXPERIMENT_DATA", currentExperimentData);
            });
            lock = false;
        } catch (error) {
            console.log(error);
        }
    });
});

io.on('connection', (socket) => {
    console.log('a user connected');
    sockets.push(socket);

    socket.on("RUN_EXPERIMENTS", async (repetitions) => {
        const queue = db.get(QUEUE);

        if (!(queue && Array.isArray(queue) && queue.length > 0)) {
            socket.emit("ERROR", "The queue is not of the correct format.");

            return;
        }
        console.log("About to loop through...");
        id = inputID;
        for (const experiment of queue) {
            const decsionRule = experiment.decisionRule;
            const percentageOfBlackTiles = experiment.percentageOfBlackTiles;
            const consensusAlgorithm = experiment.consensusAlgorithm;
            const useClassicalApproach = experiment.useClassicalApproach.toString();
            const numberOfByzantineRobots = experiment.numberOfByzantineRobots;
            const byzantineSwarmStyle = experiment.byzantineSwarmStyle;
            const numberOfRobots = experiment.numberOfRobots;

            console.log("Running experiment...");
            console.log(experiment);
            spawn("bash", [ "../scripts/clean dirs.sh", 0 ]);
            for (i = 0; i < parseInt(repetitions); i++) {
```

```
                const process = spawn("bash", ["../scripts/start_experiment.sh",0, 0,
decsionRule,                                                                consensusAlgorithm,
numberOfRobots,1,numberOfByzantineRobots,useClassicalApproach,percentageOfBlackTiles,byzantine
SwarmStyle]);
                lock = true;

                process.stdout.on('data', function (data) {
                    console.log(data.toString());
                });

                process.stderr.on('data', function (data) {
                    console.log(data.toString());
                });

                process.on('exit', function (code) {
                    console.log('child process exited with code ' + code.toString());
                });

                while (lock) {
                    await new Promise(resolve => {
                        setTimeout(() => {
                            console.log("Waiting for the current experiment to complete...");
                            resolve();
                        }, 5000);
                    });
                }

                console.log("Experiment done. Next experiment.");
            }
            id++;
        }
        sockets.forEach(socket => {
            socket.emit("EXPERIMENT_COMPLETED");
        });
    })
});

app.get("/queue", (req, res) => {
    if (db.has(QUEUE)) {
        res.send(db.get(QUEUE));

        return;
    }

    res.send([]);
});

app.post("/queue", (req, res) => {
    if (db.has(QUEUE)) {
        const queue = db.get(QUEUE);
        queue.push(req.body);
        db.set(QUEUE, queue);
    } else {
        db.set(QUEUE, [ req.body ]);
    }

    res.send();
});

app.get("/experiment-data", (req, res) => {
    if (db.has(EXPERIMENT_DATA)) {
        res.send(db.get(EXPERIMENT_DATA));

        return;
    }

    res.send([]);
});

app.delete("/queue", (req, res) => {
    db.delete(QUEUE);

    res.send();
});

app.delete("/queue/:id", (req, res) => {
    if (!db.has(QUEUE)) {
```

```
        res.status(400).send("Queue is empty");

        return;
    }

    const queue = db.get(QUEUE);
    queue.splice(req.params.id, 1);
    db.set(QUEUE, queue);

    res.send();
});

app.delete("/experiment-data", (req, res) => {
    db.delete(EXPERIMENT_DATA);

    res.send();
})

app.delete("/experiment-data/:id", (req, res) => {
    if (!db.has(EXPERIMENT DATA)) {
        res.status(400).send("Experiment data is empty");

        return;
    }

    const data = db.get(EXPERIMENT_DATA);
    data.splice(req.params.id, 1);
    db.set(EXPERIMENT_DATA, data);

    res.send();
})

app.get("/convert-to-csv", (req, res) => {
    const parser = new Parser(opts);
    const csv = parser.parse(db.get(EXPERIMENT DATA));
    res.send(csv);

})
server.listen(3000, () => {
    console.log('listening on *:3000');
});
```

7. Code of the Proof-of-Identity Algorithm

```
type PoI struct {
    config    *params.PoIConfig // Consensus engine configuration parameters
    db        ethdb.Database    // Database to store and retrieve snapshot checkpoints
    signature []byte            //Signature of the account
    publicKey *rsa.PublicKey    //public key of the swarm controller
}

type PublicKeyResponse struct {
    PublicKey string `json:"publicKey"`
}

type SignatureResponse struct {
    Signature string `json:"signature"`
}

func decodePublicKey(key string) *rsa.PublicKey {
    r := strings.NewReader(key)
    pemBytes, err := ioutil.ReadAll(r)

    if err != nil {
        panic(err)
    }

    block, _ := pem.Decode(pemBytes)
    if block == nil {
        panic(errors.New("failed to decode PEM block containing the key"))
    }

    if key, err := x509.ParsePKCS1PublicKey(block.Bytes); err == nil {
        return key
```

```
    }

    panic(err)
}

func verifySign(signature []byte, coinbase string, publicKey *rsa.PublicKey) bool {
    log.Info(coinbase)
    msgHash := sha256.New()
    msgHash.Write([]byte(coinbase))
    msgHashSum := msgHash.Sum(nil)
    return rsa.VerifyPKCS1v15(publicKey, crypto.SHA256, msgHashSum, signature) == nil
}

func getPublicKey(uri string) *rsa.PublicKey {
    resp, err := http.Get(uri + "/getPublicKey/")
    if err != nil {
        panic(err)
    }

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        panic(err)
    }

    var key PublicKeyResponse
    err = json.Unmarshal(body, &key)

    if err != nil {
        panic(err)
    }

    return decodePublicKey(key.PublicKey)
}

func (poi *PoI) SetSignature(signature []byte) {
    poi.signature = signature
}

// New creates a PoI proof-of-authority consensus engine with the initial
// signers set to the ones provided by the user.
func New(config *params.PoIConfig, db ethdb.Database) *PoI {
    // Set any missing consensus parameters to their defaults
    //log.Info("New PoI created")

    conf := *config
    if conf.NumberOfRobots == 0 {
        conf.NumberOfRobots = 10
    }

    return &PoI{
        config:    &conf,
        db:        db,
        publicKey: getPublicKey(config.SwarmControllerURI),
    }
}

// Author implements consensus.Engine, returning the header's coinbase as the
// proof-of-work verified author of the block.
func (poi *PoI) Author(header *types.Header) (common.Address, error) {
    return header.Coinbase, nil
}

func (poi *PoI) VerifyHeader(chain consensus.ChainHeaderReader, header *types.Header, seal bool)
error {

    //log.Info("Verifying Header")

    if !poi.VerifySignature(header) {
        return errors.New("signature verification failed")
    }

    log.Info("Signature Verified!")

    return nil
}
func (poi *PoI) VerifyHeaders(chain consensus.ChainHeaderReader, headers []*types.Header, seals
[]bool) (chan<- struct{}, <-chan error) {
```

```
    //log.Info("Verifying Headers")

    abort := make(chan struct{})

    results := make(chan error, len(headers))
    go func() {

        for  , header := range headers {

            err := poi.VerifyHeader(chain, header, false)

            select {

            case <-abort:

                return

            case results <- err:

            }

        }

    }()

    return abort, results

}
func (poi *PoI) VerifyUncles(chain consensus.ChainReader, block *types.Block) error {

    //log.Info("Verifying Uncles")

    if len(block.Uncles()) > 0 {
        return errors.New("uncles not allowed")
    }

    return nil
}
func (poi *PoI) VerifySignature(header *types.Header) bool {
    if verifySign(header.Extra, strings.ToLower((header.Coinbase.String())), poi.publicKey) {
        log.Info("Signature Verified!")
        return true
    }

    log.Error("Signature Verification Failed")
    return false
}
func (poi *PoI) Prepare(chain consensus.ChainHeaderReader, header *types.Header) error {

    //log.Info("Preparing the block")

    parent := chain.GetHeader(header.ParentHash, header.Number.Uint64()-1)

    if parent == nil {

        return consensus.ErrUnknownAncestor

    }
    header.Difficulty = poi.CalcDifficulty(chain, header.Time, parent)
    if poi.signature == nil {
        log.Error("the miner has no signature")
    }
    header.Extra = poi.signature
    poi.calculateTime(chain, header)

    return nil
}

func (poi *PoI) CalcDifficulty(chain consensus.ChainHeaderReader, time uint64, parent
*types.Header) *big.Int {

    return parent.Difficulty

}
```

```
func (poi *PoI) Finalize(chain consensus.ChainHeaderReader, header *types.Header, state
*state.StateDB, txs []*types.Transaction,

    uncles []*types.Header) {

    //log.Info("Finalizing the block")

    state.AddBalance(header.Coinbase, big.NewInt(100))
    header.Root = state.IntermediateRoot(chain.Config().IsEIP158(header.Number))
    header.UncleHash = types.CalcUncleHash(nil)

}

// FinalizeAndAssemble implements consensus.Engine, ensuring no uncles are set,
// nor block rewards given, and returns the final block.
func (poi *PoI) FinalizeAndAssemble(chain consensus.ChainHeaderReader, header *types.Header,
state   *state.StateDB,   txs   []*types.Transaction,   uncles   []*types.Header,   receipts
[]*types.Receipt) (*types.Block, error) {
    // Finalize block
    poi.Finalize(chain, header, state, txs, uncles)

    // Assemble and return the final block for sealing
    return types.NewBlock(header, txs, nil, receipts, trie.NewStackTrie(nil)), nil
}

func (poi *PoI) Seal(chain consensus.ChainHeaderReader, block *types.Block, results chan<-
*types.Block, stop <-chan struct{}) error {

    //log.Info("Sealing the block")

    if len(block.Transactions()) == 0 {
        //time.Sleep(15 * time.Second)
        return errors.New("sealing paused while waiting for transactions")
    }
    //time.Sleep(15 * time.Second)

    header := block.Header()

    go func() {
        select {
        case results <- block.WithSeal(header):
        case <-stop:
            return
        default:
            log.Warn("Sealing result is not read by miner", "sealhash", SealHash(header))
        }
    }()

    return nil
}

// SealHash returns the hash of a block prior to it being sealed.
func (poi *PoI) SealHash(header *types.Header) (hash common.Hash) {
    //log.Info("Sealing Hash")
    return SealHash(header)
}

// SealHash returns the hash of a block prior to it being sealed.
func SealHash(header *types.Header) (hash common.Hash) {
    hasher := sha3.NewLegacyKeccak256()
    encodeSigHeader(hasher, header)
    hasher.(gethCrypto.KeccakState).Read(hash[:])
    return hash
}

func encodeSigHeader(w io.Writer, header *types.Header) {
    enc := []interface{}{
        header.ParentHash,
        header.UncleHash,
        header.Coinbase,
        header.Root,
        header.TxHash,
        header.ReceiptHash,
        header.Bloom,
        header.Difficulty,
        header.Number,
        header.GasLimit,
```

```
        header.GasUsed,
        header.Time,
        header.Extra,
        header.MixDigest,
        header.Nonce,
    }
    if header.BaseFee != nil {
        enc = append(enc, header.BaseFee)
    }
    if err := rlp.Encode(w, enc); err != nil {
        panic("can't encode: " + err.Error())
    }
}

// APIs implements consensus.Engine, returning the user facing RPC APIs.
func (poi *PoI) APIs(chain consensus.ChainHeaderReader) []rpc.API {
    return []rpc.API{{
        Namespace: "poi",
        Version:   "1.0",
        Service:   &API{poi},
        Public:    false,
    }}
}

func (poi *PoI) Close() error {
    log.Info("Closing PoI")
    return nil
}

func (poi *PoI) calculateTime(chain consensus.ChainHeaderReader, header *types.Header) {
    var offset uint64 = 0
    alpha := 0.8

    for i := poi.config.NumberOfRobots; i > 0; i-- {
        prevHeader := chain.GetHeaderByNumber(header.Number.Uint64() - uint64(i))
        if prevHeader == nil {
            break
        }
        if prevHeader.Coinbase == header.Coinbase {
            offset += (1 / i) * poi.config.NumberOfRobots
        }
    }

    randomOffset := rand.Int63n(int64(poi.config.NumberOfRobots))

    offsetTime := uint64((alpha * float64(offset)) + ((1 - alpha) * float64(randomOffset)))

    parent := chain.GetHeaderByNumber(header.Number.Uint64() - 1)

    var blockTime uint64 = 0
    if parent.Time >= uint64(time.Now().Unix())+offsetTime {
        blockTime = parent.Time + offsetTime
    } else {
        blockTime = uint64(time.Now().Unix()) + offsetTime
    }

    header.Time = blockTime
}
```

8. End-to-end Test Results

ID	Test Case	Expected Result	Actual Result	Result Status
E2ET1	A user adds a valid experiment.	The experiment gets added to the queue.	The experiment was added to the queue.	Passed
E2ET2	A user deletes the experiment from the queue.	The experiment gets removed from the queue.	The experiment was removed from the queue.	Passed
E2ET3	A user adds two experiments to the queue and then clears the queue.	All the experiments in the queue should be removed.	All the experiments in the queue were removed.	Passed
E2ET4	A user adds an experiment, sets repetitions to 10, and runs the experiment.	The results of the 10 experiments should be shown on the table live.	The results of the 10 experiments were shown on the table live.	Passed
E2ET5	A user removes the first experiment data from the table.	The experiment data gets removed from the table.	The experiment data was removed from the table.	Passed
E2ET6	A user clicks on the "Download CSV" button.	The experiment data gets downloaded as a CSV file.	The experiment data was downloaded in the CSV format.	Passed
E2ET7	A user tries to run experiments without any experiments in the queue.	The "RUN EXPERIMENTS" button remains disabled.	The "RUN EXPERIMENTS" button remained disabled.	Passed
E2ET7	A user clears the table.	All the results on the table get removed.	All the results were removed.	Passed
E2ET8	A user tries to switch color mode.	The UI switches to the dark/light mode based on the current mode.	The UI switched from dark to light and light to dark.	Passed

Table 51 The results of the end-to-end tests

9. Requirement Fulfillment Validation

Requirement ID	Test Case ID/Test Results	Test Status
FR1	FT1	Passed
FR2	FT1	Passed
FR3	FT2	Passed
FR4	FT2	Passed
FR5	FT3	Passed
FR6	FT2, FT3	Passed
FR7	FT5	Passed
FR8	FT5	Passed
FR9	FT7	Passed
FR10	Table 34	Passed
NFR1	Table 34	Passed
NFR2	Table 32	Passed
NFR3	ST1, ST2, ST3	Passed
NFR4	FT1	Passed
NFR5	Figure 15	Passed

Table 52 Validation of requirement fulfillment

10. Code

The code produced during this research work can be found in this GitHub organization:

https://github.com/PoI-Research