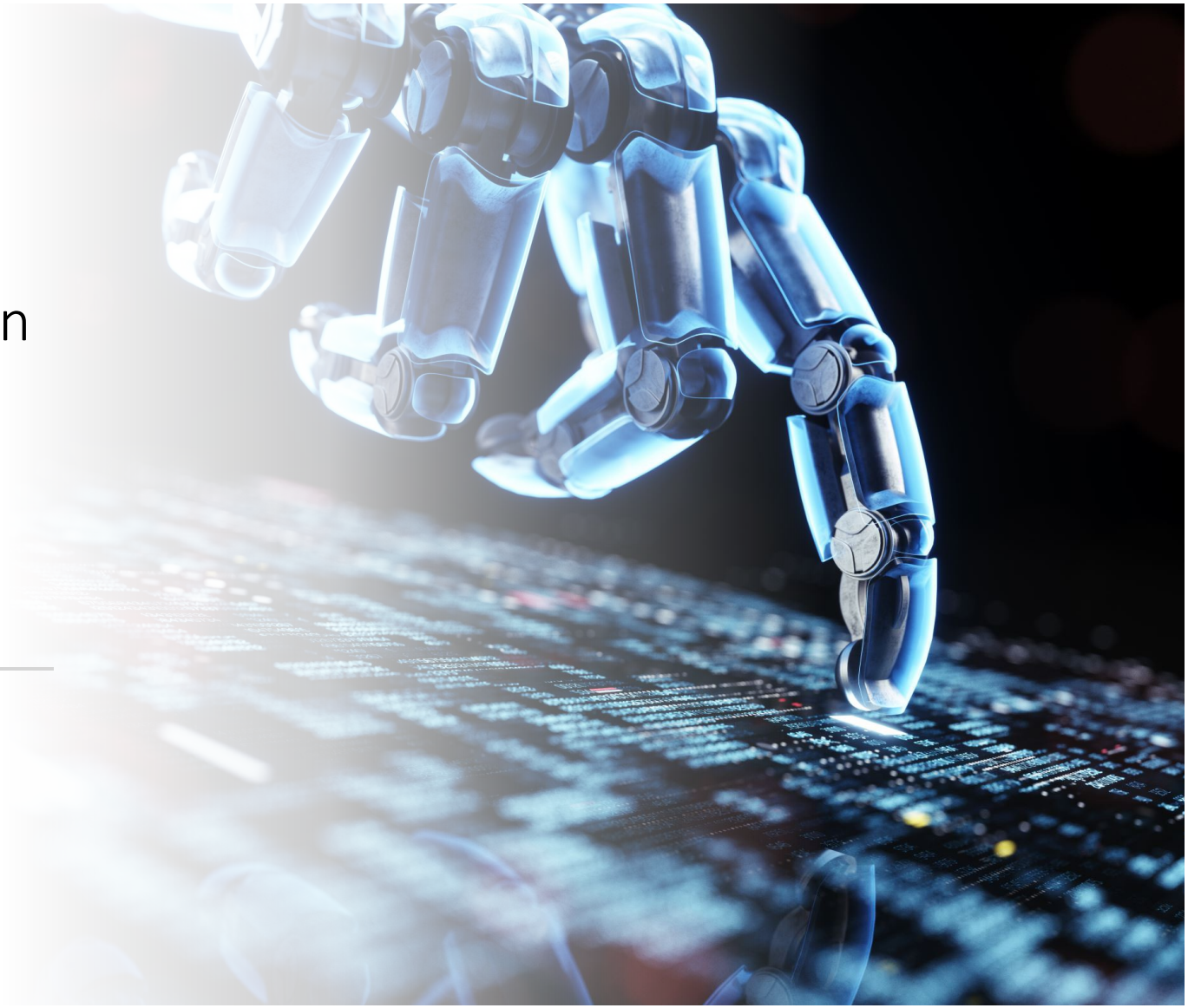
Managing Byzantine Robots via Blockchain Technology in a Swarm Robotics Collective Decision Making Scenario



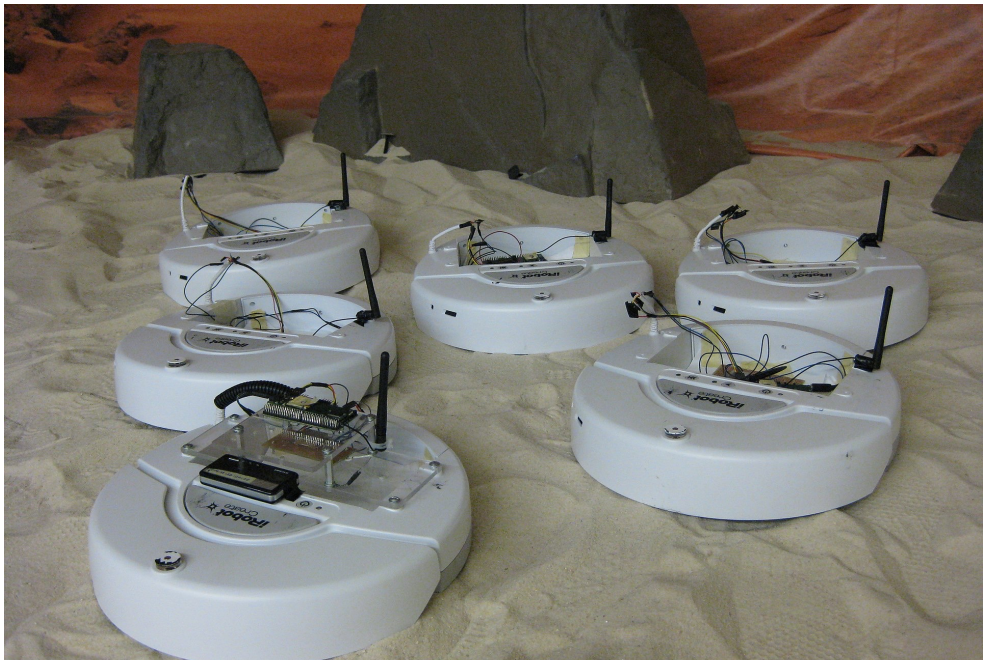
Presenter: Ye Yuan

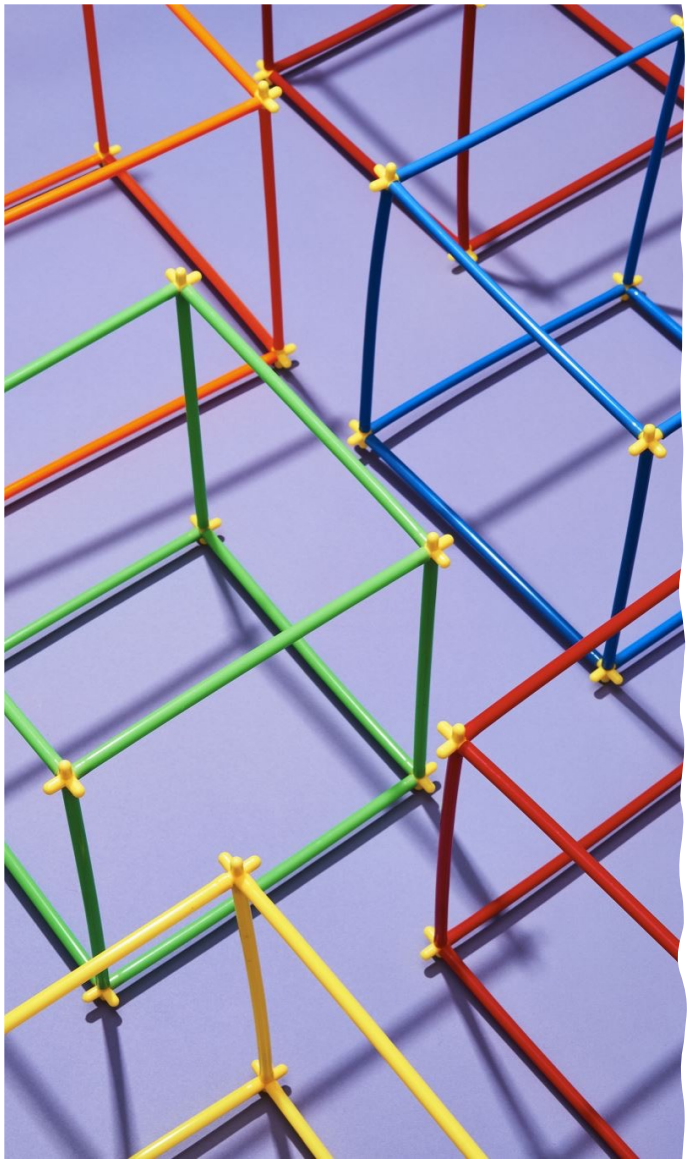
2023.02.11

Swarm Robotics

- Swarm robotics is an approach to collective robotics that takes inspiration from the self-organized behaviors of social animals. Through simple rules and local interactions, swarm robotics aims at designing robust, scalable, and flexible collective behaviors for coordinating large numbers of robots.
- One of the most promising uses of swarm robotics is in search and rescue missions. Swarms of robots of different sizes could be sent to places that rescue workers can't reach safely to explore the unknown environment and solve complex mazes via onboard sensors.[9] On the other hand, swarm robotics can be suited to tasks that demand cheap designs, such as mining or agricultural shepherding.
- More controversially, swarms of military robots can form an autonomous army. U.S. Naval forces have tested a swarm of autonomous boats that can steer and take offensive actions by themselves. The unmanned boats can be fitted with any kit to deter and destroy enemy vessels.

Swarm Robotics in laboratory settings





Paper Background

- Volker Strobel, Eduardo Castelló Ferrer, and Marco Dorigo
- Université libre de Bruxelles and MIT Media Lab
- Accepted by International Conference on Autonomous Agents and Multiagent Systems (AAMAS)

Catalog

- Abstract
- Introduction
- Related Work
- Fundamentals of Blockchain Technology
- Methods
- Experiments
- Discussion
- Conclusion and Future Work
- Acknowledgement

Abstract

- Introduced the limitation of the current research in this field
- Point out the problem
- Summarize the proposed approach in their work (emphasize the novelty)
- Summarize the experiments configurations
- Conclusion

ABSTRACT

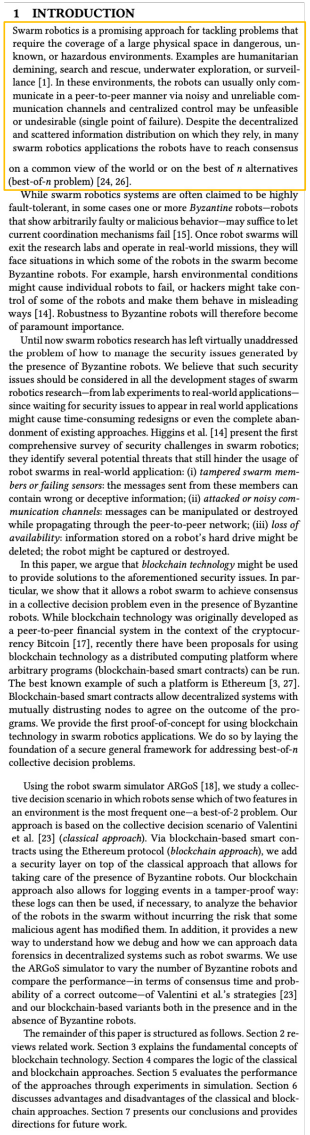
While swarm robotics systems are often claimed to be highly fault-tolerant, so far research has limited its attention to safe laboratory settings and has virtually ignored security issues in the presence of Byzantine robots—i.e., robots with arbitrarily faulty or malicious behavior. However, in many applications one or more Byzantine robots may suffice to let current swarm coordination mechanisms fail with unpredictable or disastrous outcomes. In this paper, we provide a proof-of-concept for managing security issues in swarm robotics systems via blockchain technology. Our approach uses decentralized programs executed via blockchain technology (blockchain-based smart contracts) to establish secure swarm coordination mechanisms and to identify and exclude Byzantine swarm members. We studied the performance of our blockchain-based approach in a collective decision-making scenario both in the presence and absence of Byzantine robots and compared our results to those obtained with an existing collective decision approach. The results show a clear advantage of the blockchain approach when Byzantine robots are part of the swarm.

Introduction

- Examples of Application
- Why need consensus in Swarm Robotics

1 INTRODUCTION

Swarm robotics is a promising approach for tackling problems that require the coverage of a large physical space in dangerous, unknown, or hazardous environments. Examples are humanitarian demining, search and rescue, underwater exploration, or surveillance [1]. In these environments, the robots can usually only communicate in a peer-to-peer manner via noisy and unreliable communication channels and centralized control may be unfeasible or undesirable (single point of failure). Despite the decentralized and scattered information distribution on which they rely, in many swarm robotics applications the robots have to reach consensus on a common view of the world or on the best of n alternatives (best-of-n problem) [24, 26].



1 INTRODUCTION

Swarm robotics is a promising approach for tackling problems that require the coverage of a large physical space in dangerous, unknown, or hazardous environments. Examples are humanitarian demining, search and rescue, underwater exploration, or surveillance [1]. In these environments, the robots can usually only communicate in a peer-to-peer manner via noisy and unreliable communication channels and centralized control may be unfeasible or undesirable (single point of failure). Despite the decentralized and scattered information distribution on which they rely, in many swarm robotics applications the robots have to reach consensus on a common view of the world or on the best of n alternatives (best-of-n problem) [24, 26].

While swarm robotics systems are often claimed to be highly fault-tolerant, in some cases one or more *Byzantine* robots—robots that show arbitrarily faulty or malicious behavior—may suffice to let current coordination mechanisms fail [15]. Once robot swarms will exit the research labs and operate in real-world missions, they will face situations in which some of the robots in the swarm become Byzantine robots. For example, harsh environmental conditions might cause individual robots to fail, or hackers might take control of some of the robots and make them behave in misleading ways [14]. Robustness to Byzantine robots will therefore become of paramount importance.

Until now swarm robotics research has left virtually unaddressed the problem of how to manage the security issues generated by the presence of Byzantine robots. We believe that such security issues should be considered in all the development stages of swarm robotics research—from lab experiments to real-world applications—since waiting for security issues to appear in real world applications might cause time-consuming redesigns or even the complete abandonment of existing approaches. Higgins et al. [14] present the first comprehensive survey of security challenges in swarm robotics; they identify several potential threats that still hinder the usage of robot swarms in real-world application: (i) *tampered swarm members or failing sensors*: the messages sent from these members can contain wrong or deceptive information; (ii) *attacked or noisy communication channels*: messages can be manipulated or destroyed while propagating through the peer-to-peer network; (iii) *loss of availability*: information stored on a robot's hard drive might be deleted, the robot might be captured or destroyed.

In this paper, we argue that *blockchain technology* might be used to provide solutions to the aforementioned security issues. In particular, we show that it allows a robot swarm to achieve consensus in a collective decision problem even in the presence of Byzantine robots. While blockchain technology was originally developed as a peer-to-peer financial system in the context of the cryptocurrency Bitcoin [17], recently there have been proposals for using blockchain technology as a distributed computing platform where arbitrary programs (blockchain-based smart contracts) can be run. The best known example of such a platform is Ethereum [3, 27]. Blockchain-based smart contracts allow decentralized systems with mutually distrusting nodes to agree on the outcome of the programs. We provide the first proof-of-concept for using blockchain technology in swarm robotics applications. We do so by laying the foundation of a secure general framework for addressing best-of-n collective decision problems.

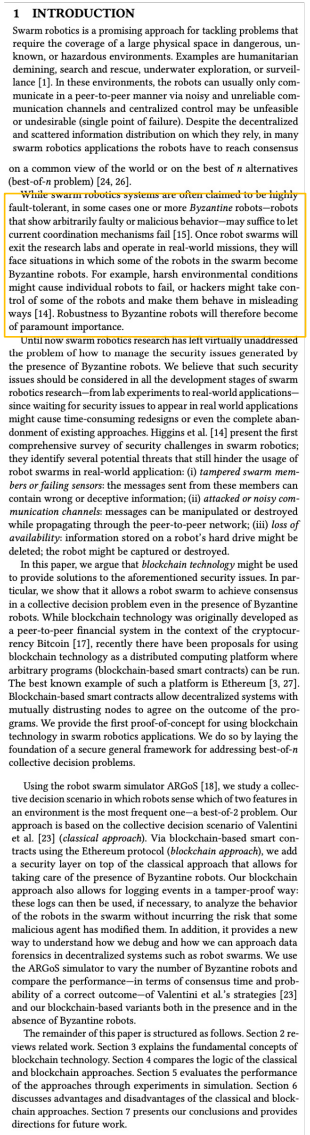
Using the robot swarm simulator ARGoS [18], we study a collective decision scenario in which robots sense which of two features in an environment is the most frequent one—a best-of-2 problem. Our approach is based on the collective decision scenario of Valentini et al. [23] (*classical approach*). Via blockchain-based smart contracts using the Ethereum protocol (*blockchain approach*), we add a security layer on top of the classical approach that allows for taking care of the presence of Byzantine robots. Our blockchain approach also allows for logging events in a tamper-proof way: these logs can then be used, if necessary, to analyze the behavior of the robots in the swarm without incurring the risk that some malicious agent has modified them. In addition, it provides a new way to understand how we debug and how we can approach data forensics in decentralized systems such as robot swarms. We use the ARGoS simulator to vary the number of Byzantine robots and compare the performance—in terms of consensus time and probability of a correct outcome—of Valentini et al.'s strategies [23] and our blockchain-based variants both in the presence and in the absence of Byzantine robots.

The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 explains the fundamental concepts of blockchain technology. Section 4 compares the logic of the classical and blockchain approaches. Section 5 evaluates the performance of the approaches through experiments in simulation. Section 6 discusses advantages and disadvantages of the classical and blockchain approaches. Section 7 presents our conclusions and provides directions for future work.

Introduction

- Use cases
- Show why our research is important

While swarm robotics systems are often claimed to be highly fault-tolerant, in some cases one or more *Byzantine* robots—robots that show arbitrarily faulty or malicious behavior—may suffice to let current coordination mechanisms fail [15]. Once robot swarms will exit the research labs and operate in real-world missions, they will face situations in which some of the robots in the swarm become Byzantine robots. For example, harsh environmental conditions might cause individual robots to fail, or hackers might take control of some of the robots and make them behave in misleading ways [14]. Robustness to Byzantine robots will therefore become of paramount importance.



1 INTRODUCTION

Swarm robotics is a promising approach for tackling problems that require the coverage of a large physical space in dangerous, unknown, or hazardous environments. Examples are humanitarian demining, search and rescue, underwater exploration, or surveillance [1]. In these environments, the robots can usually only communicate in a peer-to-peer manner via noisy and unreliable communication channels and centralized control may be unfeasible or undesirable (single point of failure). Despite the decentralized and scattered information distribution on which they rely, in many swarm robotics applications the robots have to reach consensus on a common view of the world or on the best of n alternatives (best-of-n problem) [24, 26].

While swarm robotics systems are often claimed to be highly fault-tolerant, in some cases one or more *Byzantine* robots—robots that show arbitrarily faulty or malicious behavior—may suffice to let current coordination mechanisms fail [15]. Once robot swarms will exit the research labs and operate in real-world missions, they will face situations in which some of the robots in the swarm become Byzantine robots. For example, harsh environmental conditions might cause individual robots to fail, or hackers might take control of some of the robots and make them behave in misleading ways [14]. Robustness to Byzantine robots will therefore become of paramount importance.

Until now swarm robotics research has left virtually unaddressed the problem of how to manage the security issues generated by the presence of Byzantine robots. We believe that such security issues should be considered in all the development stages of swarm robotics research—from lab experiments to real-world applications—since waiting for security issues to appear in real world applications might cause time-consuming redesigns or even the complete abandonment of existing approaches. Higgins et al. [14] present the first comprehensive survey of security challenges in swarm robotics; they identify several potential threats that still hinder the usage of robot swarms in real-world application: (i) *tampered swarm members or failing sensors*: the messages sent from these members can contain wrong or deceptive information; (ii) *attacked or noisy communication channels*: messages can be manipulated or destroyed while propagating through the peer-to-peer network; (iii) *loss of availability*: information stored on a robot's hard drive might be deleted, the robot might be captured or destroyed.

In this paper, we argue that *blockchain technology* might be used to provide solutions to the aforementioned security issues. In particular, we show that it allows a robot swarm to achieve consensus in a collective decision problem even in the presence of Byzantine robots. While blockchain technology was originally developed as a peer-to-peer financial system in the context of the cryptocurrency Bitcoin [17], recently there have been proposals for using blockchain technology as a distributed computing platform where arbitrary programs (blockchain-based smart contracts) can be run. The best known example of such a platform is Ethereum [3, 27]. Blockchain-based smart contracts allow decentralized systems with mutually distrusting nodes to agree on the outcome of the programs. We provide the first proof-of-concept for using blockchain technology in swarm robotics applications. We do so by laying the foundation of a secure general framework for addressing best-of-n collective decision problems.

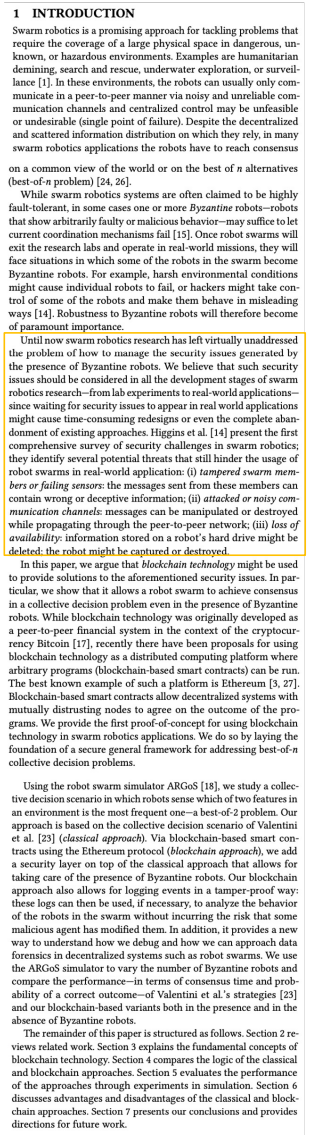
Using the robot swarm simulator ARGoS [18], we study a collective decision scenario in which robots sense which of two features in an environment is the most frequent one—a best-of-2 problem. Our approach is based on the collective decision scenario of Valentini et al. [23] (*classical approach*). Via blockchain-based smart contracts using the Ethereum protocol (*blockchain approach*), we add a security layer on top of the classical approach that allows for taking care of the presence of Byzantine robots. Our blockchain approach also allows for logging events in a tamper-proof way: these logs can then be used, if necessary, to analyze the behavior of the robots in the swarm without incurring the risk that some malicious agent has modified them. In addition, it provides a new way to understand how we debug and how we can approach data forensics in decentralized systems such as robot swarms. We use the ARGoS simulator to vary the number of Byzantine robots and compare the performance—in terms of consensus time and probability of a correct outcome—of Valentini et al.'s strategies [23] and our blockchain-based variants both in the presence and in the absence of Byzantine robots.

The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 explains the fundamental concepts of blockchain technology. Section 4 compares the logic of the classical and blockchain approaches. Section 5 evaluates the performance of the approaches through experiments in simulation. Section 6 discusses advantages and disadvantages of the classical and blockchain approaches. Section 7 presents our conclusions and provides directions for future work.

Introduction

- Emphasize the importance of this research
- Introduce three potential threats mentioned in other works.

Until now swarm robotics research has left virtually unaddressed the problem of how to manage the security issues generated by the presence of Byzantine robots. We believe that such security issues should be considered in all the development stages of swarm robotics research—from lab experiments to real-world applications—since waiting for security issues to appear in real world applications might cause time-consuming redesigns or even the complete abandonment of existing approaches. Higgins et al. [14] present the first comprehensive survey of security challenges in swarm robotics; they identify several potential threats that still hinder the usage of robot swarms in real-world application: (i) *tampered swarm members or failing sensors*: the messages sent from these members can contain wrong or deceptive information; (ii) *attacked or noisy communication channels*: messages can be manipulated or destroyed while propagating through the peer-to-peer network; (iii) *loss of availability*: information stored on a robot's hard drive might be deleted; the robot might be captured or destroyed.



1 INTRODUCTION

Swarm robotics is a promising approach for tackling problems that require the coverage of a large physical space in dangerous, unknown, or hazardous environments. Examples are humanitarian demining, search and rescue, underwater exploration, or surveillance [1]. In these environments, the robots can usually only communicate in a peer-to-peer manner via noisy and unreliable communication channels and centralized control may be unfeasible or undesirable (single point of failure). Despite the decentralized and scattered information distribution on which they rely, in many swarm robotics applications the robots have to reach consensus on a common view of the world or on the best of n alternatives (best-of-n problem) [24, 26].

While swarm robotics systems are often claimed to be highly fault-tolerant, in some cases one or more *Byzantine* robots—robots that show arbitrarily faulty or malicious behavior—may suffice to let current coordination mechanisms fail [15]. Once robot swarms will exit the research labs and operate in real-world missions, they will face situations in which some of the robots in the swarm become Byzantine robots. For example, harsh environmental conditions might cause individual robots to fail, or hackers might take control of some of the robots and make them behave in misleading ways [14]. Robustness to Byzantine robots will therefore become of paramount importance.

Until now swarm robotics research has left virtually unaddressed the problem of how to manage the security issues generated by the presence of Byzantine robots. We believe that such security issues should be considered in all the development stages of swarm robotics research—from lab experiments to real-world applications—since waiting for security issues to appear in real world applications might cause time-consuming redesigns or even the complete abandonment of existing approaches. Higgins et al. [14] present the first comprehensive survey of security challenges in swarm robotics; they identify several potential threats that still hinder the usage of robot swarms in real-world application: (i) *tampered swarm members or failing sensors*: the messages sent from these members can contain wrong or deceptive information; (ii) *attacked or noisy communication channels*: messages can be manipulated or destroyed while propagating through the peer-to-peer network; (iii) *loss of availability*: information stored on a robot's hard drive might be deleted; the robot might be captured or destroyed.

In this paper, we argue that *blockchain technology* might be used to provide solutions to the aforementioned security issues. In particular, we show that it allows a robot swarm to achieve consensus in a collective decision problem even in the presence of Byzantine robots. While blockchain technology was originally developed as a peer-to-peer financial system in the context of the cryptocurrency Bitcoin [17], recently there have been proposals for using blockchain technology as a distributed computing platform where arbitrary programs (blockchain-based smart contracts) can be run. The best known example of such a platform is Ethereum [3, 27]. Blockchain-based smart contracts allow decentralized systems with mutually distrusting nodes to agree on the outcome of the programs. We provide the first proof-of-concept for using blockchain technology in swarm robotics applications. We do so by laying the foundation of a secure general framework for addressing best-of-n collective decision problems.

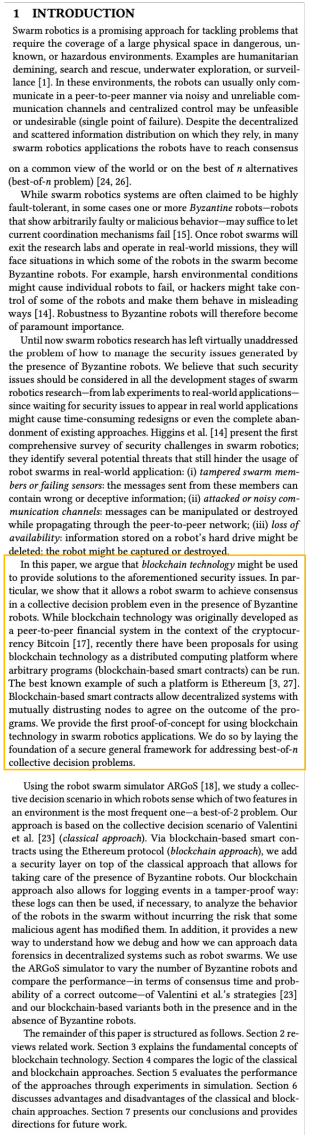
Using the robot swarm simulator ARGoS [18], we study a collective decision scenario in which robots sense which of two features in an environment is the most frequent one—a best-of-2 problem. Our approach is based on the collective decision scenario of Valentini et al. [23] (*classical approach*). Via blockchain-based smart contracts using the Ethereum protocol (*blockchain approach*), we add a security layer on top of the classical approach that allows for taking care of the presence of Byzantine robots. Our blockchain approach also allows for logging events in a tamper-proof way: these logs can then be used, if necessary, to analyze the behavior of the robots in the swarm without incurring the risk that some malicious agent has modified them. In addition, it provides a new way to understand how we debug and how we can approach data forensics in decentralized systems such as robot swarms. We use the ARGoS simulator to vary the number of Byzantine robots and compare the performance—in terms of consensus time and probability of a correct outcome—of Valentini et al.'s strategies [23] and our blockchain-based variants both in the presence and in the absence of Byzantine robots.

The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 explains the fundamental concepts of blockchain technology. Section 4 compares the logic of the classical and blockchain approaches. Section 5 evaluates the performance of the approaches through experiments in simulation. Section 6 discusses advantages and disadvantages of the classical and blockchain approaches. Section 7 presents our conclusions and provides directions for future work.

Introduction

- Introduce the proposed solution
- Summarize the blockchain tech
  (which would be helpful for reviewers)
- Again, stress the proposed solution

In this paper, we argue that *blockchain technology* might be used to provide solutions to the aforementioned security issues. In particular, we show that it allows a robot swarm to achieve consensus in a collective decision problem even in the presence of Byzantine robots. While blockchain technology was originally developed as a peer-to-peer financial system in the context of the cryptocurrency Bitcoin [17], recently there have been proposals for using blockchain technology as a distributed computing platform where arbitrary programs (blockchain-based smart contracts) can be run. The best known example of such a platform is Ethereum [3, 27]. Blockchain-based smart contracts allow decentralized systems with mutually distrusting nodes to agree on the outcome of the programs. We provide the first proof-of-concept for using blockchain technology in swarm robotics applications. We do so by laying the foundation of a secure general framework for addressing best-of-n collective decision problems.



1 INTRODUCTION

Swarm robotics is a promising approach for tackling problems that require the coverage of a large physical space in dangerous, unknown, or hazardous environments. Examples are humanitarian demining, search and rescue, underwater exploration, or surveillance [1]. In these environments, the robots can usually only communicate in a peer-to-peer manner via noisy and unreliable communication channels and centralized control may be unfeasible or undesirable (single point of failure). Despite the decentralized and scattered information distribution on which they rely, in many swarm robotics applications the robots have to reach consensus on a common view of the world or on the best of n alternatives (best-of-n problem) [24, 26].

While swarm robotics systems are often claimed to be highly fault-tolerant, in some cases one or more *Byzantine* robots—robots that show arbitrarily faulty or malicious behavior—may suffice to let current coordination mechanisms fail [15]. Once robot swarms will exit the research labs and operate in real-world missions, they will face situations in which some of the robots in the swarm become Byzantine robots. For example, harsh environmental conditions might cause individual robots to fail, or hackers might take control of some of the robots and make them behave in misleading ways [14]. Robustness to Byzantine robots will therefore become of paramount importance.

Until now swarm robotics research has left virtually unaddressed the problem of how to manage the security issues generated by the presence of Byzantine robots. We believe that such security issues should be considered in all the development stages of swarm robotics research—from lab experiments to real-world applications—since waiting for security issues to appear in real world applications might cause time-consuming redesigns or even the complete abandonment of existing approaches. Higgins et al. [14] present the first comprehensive survey of security challenges in swarm robotics; they identify several potential threats that still hinder the usage of robot swarms in real-world application: (i) *tampered swarm members or failing sensors*: the messages sent from these members can contain wrong or deceptive information; (ii) *attacked or noisy communication channels*: messages can be manipulated or destroyed while propagating through the peer-to-peer network; (iii) *loss of availability*: information stored on a robot's hard drive might be deleted, the robot might be captured or destroyed.

In this paper, we argue that *blockchain technology* might be used to provide solutions to the aforementioned security issues. In particular, we show that it allows a robot swarm to achieve consensus in a collective decision problem even in the presence of Byzantine robots. While blockchain technology was originally developed as a peer-to-peer financial system in the context of the cryptocurrency Bitcoin [17], recently there have been proposals for using blockchain technology as a distributed computing platform where arbitrary programs (blockchain-based smart contracts) can be run. The best known example of such a platform is Ethereum [3, 27]. Blockchain-based smart contracts allow decentralized systems with mutually distrusting nodes to agree on the outcome of the programs. We provide the first proof-of-concept for using blockchain technology in swarm robotics applications. We do so by laying the foundation of a secure general framework for addressing best-of-n collective decision problems.

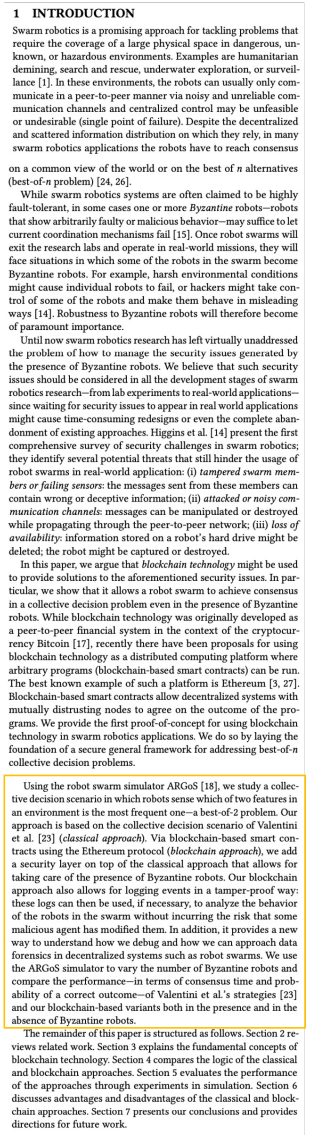
Using the robot swarm simulator ARGoS [18], we study a collective decision scenario in which robots sense which of two features in an environment is the most frequent one—a best-of-2 problem. Our approach is based on the collective decision scenario of Valentini et al. [23] (*classical approach*). Via blockchain-based smart contracts using the Ethereum protocol (*blockchain approach*), we add a security layer on top of the classical approach that allows for taking care of the presence of Byzantine robots. Our blockchain approach also allows for logging events in a tamper-proof way: these logs can then be used, if necessary, to analyze the behavior of the robots in the swarm without incurring the risk that some malicious agent has modified them. In addition, it provides a new way to understand how we debug and how we can approach data forensics in decentralized systems such as robot swarms. We use the ARGoS simulator to vary the number of Byzantine robots and compare the performance—in terms of consensus time and probability of a correct outcome—of Valentini et al.'s strategies [23] and our blockchain-based variants both in the presence and in the absence of Byzantine robots.

The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 explains the fundamental concepts of blockchain technology. Section 4 compares the logic of the classical and blockchain approaches. Section 5 evaluates the performance of the approaches through experiments in simulation. Section 6 discusses advantages and disadvantages of the classical and blockchain approaches. Section 7 presents our conclusions and provides directions for future work.

Introduction

- Introduce experiments settings
- Summarize the proposed approach
- Three main contributions
- Metrics used

Using the robot swarm simulator ARGoS [18], we study a collective decision scenario in which robots sense which of two features in an environment is the most frequent one—a best-of-2 problem. Our approach is based on the collective decision scenario of Valentini et al. [23] (*classical approach*). Via blockchain-based smart contracts using the Ethereum protocol (*blockchain approach*), we add a security layer on top of the classical approach that allows for taking care of the presence of Byzantine robots. Our blockchain approach also allows for logging events in a tamper-proof way: these logs can then be used, if necessary, to analyze the behavior of the robots in the swarm without incurring the risk that some malicious agent has modified them. In addition, it provides a new way to understand how we debug and how we can approach data forensics in decentralized systems such as robot swarms. We use the ARGoS simulator to vary the number of Byzantine robots and compare the performance—in terms of consensus time and probability of a correct outcome—of Valentini et al.'s strategies [23] and our blockchain-based variants both in the presence and in the absence of Byzantine robots.



1 INTRODUCTION

Swarm robotics is a promising approach for tackling problems that require the coverage of a large physical space in dangerous, unknown, or hazardous environments. Examples are humanitarian demining, search and rescue, underwater exploration, or surveillance [1]. In these environments, the robots can usually only communicate in a peer-to-peer manner via noisy and unreliable communication channels and centralized control may be unfeasible or undesirable (single point of failure). Despite the decentralized and scattered information distribution on which they rely, in many swarm robotics applications the robots have to reach consensus on a common view of the world or on the best of n alternatives (best-of-n problem) [24, 26].

While swarm robotics systems are often claimed to be highly fault-tolerant, in some cases one or more *Byzantine* robots—robots that show arbitrarily faulty or malicious behavior—may suffice to let current coordination mechanisms fail [15]. Once robot swarms will exit the research labs and operate in real-world missions, they will face situations in which some of the robots in the swarm become Byzantine robots. For example, harsh environmental conditions might cause individual robots to fail, or hackers might take control of some of the robots and make them behave in misleading ways [14]. Robustness to Byzantine robots will therefore become of paramount importance.

Until now swarm robotics research has left virtually unaddressed the problem of how to manage the security issues generated by the presence of Byzantine robots. We believe that such security issues should be considered in all the development stages of swarm robotics research—from lab experiments to real-world applications—since waiting for security issues to appear in real world applications might cause time-consuming redesigns or even the complete abandonment of existing approaches. Higgins et al. [14] present the first comprehensive survey of security challenges in swarm robotics; they identify several potential threats that still hinder the usage of robot swarms in real-world application: (i) *tampered swarm members or failing sensors*: the messages sent from these members can contain wrong or deceptive information; (ii) *attacked or noisy communication channels*: messages can be manipulated or destroyed while propagating through the peer-to-peer network; (iii) *loss of availability*: information stored on a robot's hard drive might be deleted, the robot might be captured or destroyed.

In this paper, we argue that *blockchain technology* might be used to provide solutions to the aforementioned security issues. In particular, we show that it allows a robot swarm to achieve consensus in a collective decision problem even in the presence of Byzantine robots. While blockchain technology was originally developed as a peer-to-peer financial system in the context of the cryptocurrency Bitcoin [17], recently there have been proposals for using blockchain technology as a distributed computing platform where arbitrary programs (blockchain-based smart contracts) can be run. The best known example of such a platform is Ethereum [3, 27]. Blockchain-based smart contracts allow decentralized systems with mutually distrusting nodes to agree on the outcome of the programs. We provide the first proof-of-concept for using blockchain technology in swarm robotics applications. We do so by laying the foundation of a secure general framework for addressing best-of-n collective decision problems.

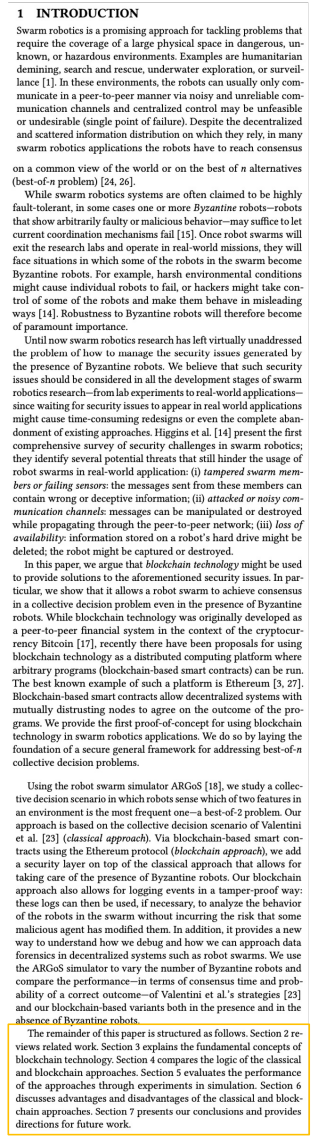
Using the robot swarm simulator ARGoS [18], we study a collective decision scenario in which robots sense which of two features in an environment is the most frequent one—a best-of-2 problem. Our approach is based on the collective decision scenario of Valentini et al. [23] (*classical approach*). Via blockchain-based smart contracts using the Ethereum protocol (*blockchain approach*), we add a security layer on top of the classical approach that allows for taking care of the presence of Byzantine robots. Our blockchain approach also allows for logging events in a tamper-proof way: these logs can then be used, if necessary, to analyze the behavior of the robots in the swarm without incurring the risk that some malicious agent has modified them. In addition, it provides a new way to understand how we debug and how we can approach data forensics in decentralized systems such as robot swarms. We use the ARGoS simulator to vary the number of Byzantine robots and compare the performance—in terms of consensus time and probability of a correct outcome—of Valentini et al.'s strategies [23] and our blockchain-based variants both in the presence and in the absence of Byzantine robots.

The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 explains the fundamental concepts of blockchain technology. Section 4 compares the logic of the classical and blockchain approaches. Section 5 evaluates the performance of the approaches through experiments in simulation. Section 6 discusses advantages and disadvantages of the classical and blockchain approaches. Section 7 presents our conclusions and provides directions for future work.

Introduction

- Following structure

The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 explains the fundamental concepts of blockchain technology. Section 4 compares the logic of the classical and blockchain approaches. Section 5 evaluates the performance of the approaches through experiments in simulation. Section 6 discusses advantages and disadvantages of the classical and blockchain approaches. Section 7 presents our conclusions and provides directions for future work.



1 INTRODUCTION

Swarm robotics is a promising approach for tackling problems that require the coverage of a large physical space in dangerous, unknown, or hazardous environments. Examples are humanitarian demining, search and rescue, underwater exploration, or surveillance [1]. In these environments, the robots can usually only communicate in a peer-to-peer manner via noisy and unreliable communication channels and centralized control may be unfeasible or undesirable (single point of failure). Despite the decentralized and scattered information distribution on which they rely, in many swarm robotics applications the robots have to reach consensus on a common view of the world or on the best of n alternatives (best-of-n problem) [24, 26].

While swarm robotics systems are often claimed to be highly fault-tolerant, in some cases one or more *Byzantine* robots—robots that show arbitrarily faulty or malicious behavior—may suffice to let current coordination mechanisms fail [15]. Once robot swarms will exit the research labs and operate in real-world missions, they will face situations in which some of the robots in the swarm become Byzantine robots. For example, harsh environmental conditions might cause individual robots to fail, or hackers might take control of some of the robots and make them behave in misleading ways [14]. Robustness to Byzantine robots will therefore become of paramount importance.

Until now swarm robotics research has left virtually unaddressed the problem of how to manage the security issues generated by the presence of Byzantine robots. We believe that such security issues should be considered in all the development stages of swarm robotics research—from lab experiments to real-world applications—since waiting for security issues to appear in real world applications might cause time-consuming redesigns or even the complete abandonment of existing approaches. Higgins et al. [14] present the first comprehensive survey of security challenges in swarm robotics; they identify several potential threats that still hinder the usage of robot swarms in real-world application: (i) *tampered swarm members or failing sensors*: the messages sent from these members can contain wrong or deceptive information; (ii) *attacked or noisy communication channels*: messages can be manipulated or destroyed while propagating through the peer-to-peer network; (iii) *loss of availability*: information stored on a robot's hard drive might be deleted, the robot might be captured or destroyed.

In this paper, we argue that *blockchain technology* might be used to provide solutions to the aforementioned security issues. In particular, we show that it allows a robot swarm to achieve consensus in a collective decision problem even in the presence of Byzantine robots. While blockchain technology was originally developed as a peer-to-peer financial system in the context of the cryptocurrency Bitcoin [17], recently there have been proposals for using blockchain technology as a distributed computing platform where arbitrary programs (blockchain-based smart contracts) can be run. The best known example of such a platform is Ethereum [3, 27]. Blockchain-based smart contracts allow decentralized systems with mutually distrusting nodes to agree on the outcome of the programs. We provide the first proof-of-concept for using blockchain technology in swarm robotics applications. We do so by laying the foundation of a secure general framework for addressing best-of-n collective decision problems.

Using the robot swarm simulator ARGoS [18], we study a collective decision scenario in which robots sense which of two features in an environment is the most frequent one—a best-of-2 problem. Our approach is based on the collective decision scenario of Valentini et al. [23] (*classical approach*). Via blockchain-based smart contracts using the Ethereum protocol (*blockchain approach*), we add a security layer on top of the classical approach that allows for taking care of the presence of Byzantine robots. Our blockchain approach also allows for logging events in a tamper-proof way: these logs can then be used, if necessary, to analyze the behavior of the robots in the swarm without incurring the risk that some malicious agent has modified them. In addition, it provides a new way to understand how we debug and how we can approach data forensics in decentralized systems such as robot swarms. We use the ARGoS simulator to vary the number of Byzantine robots and compare the performance—in terms of consensus time and probability of a correct outcome—of Valentini et al.'s strategies [23] and our blockchain-based variants both in the presence and in the absence of Byzantine robots.

The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 explains the fundamental concepts of blockchain technology. Section 4 compares the logic of the classical and blockchain approaches. Section 5 evaluates the performance of the approaches through experiments in simulation. Section 6 discusses advantages and disadvantages of the classical and blockchain approaches. Section 7 presents our conclusions and provides directions for future work.

Related Work

- Two classes (introduction)
- Introduce the scenario used in this research

2 RELATED WORK

Many studies (e.g., [2, 12, 13, 16, 19, 20, 25]) have addressed collective decision-making in robot swarms, a key task for the advancement of swarm robotics [7]. Collective decision-making tasks can be divided into two sub-classes: task allocation and consensus achievement [1]. In task allocation, the goal of the swarm is to maximize the overall swarm performance by assigning the members to different tasks. In consensus achievement, the goal of the swarm is to agree upon the best among a set of alternatives. The scenario that we use in this paper is based on the work of Valentini et al. [23], who describe and evaluate a collective decision scenario where robots have to reach consensus on the most frequent tile color in an environment in which the floor is covered with black and white tiles.

2 RELATED WORK

Many studies (e.g., [2, 12, 13, 16, 19, 20, 25]) have addressed collective decision-making in robot swarms, a key task for the advancement of swarm robotics [7]. Collective decision-making tasks can be divided into two sub-classes: task allocation and consensus achievement [1]. In task allocation, the goal of the swarm is to maximize the overall swarm performance by assigning the members to different tasks. In consensus achievement, the goal of the swarm is to agree upon the best among a set of alternatives. The scenario that we use in this paper is based on the work of Valentini et al. [23], who describe and evaluate a collective decision scenario where robots have to reach consensus on the most frequent tile color in an environment in which the floor is covered with black and white tiles.

Robot swarms are often assumed to be fault-tolerant *by design* [15], therefore, the explicit detection of faults was initially given little attention in swarm robotics research. Early security research focused on fault detection in the presence of *defective* robots [5, 6]. More recently, the explicit modeling of *malicious* robots has attracted more attention: [22] gives an overview of robot swarms' robustness to different attacker strategies in a cooperative navigation experiment and [28] presents a reputation management system that assigns a dynamic trust level to robots to identify malicious entities. In [10], a lightweight algorithm for detecting Sybil attacks via physical properties of wireless signals is developed. Connectivity requirements for achieving consensus in the presence of malicious robots are studied in [11]. [21] presents a resilient consensus protocol for dynamic agents whose network topology changes over time. However, so far, no secure general frameworks exist to cope with security issues in a fully decentralized way. Currently, blockchain technology mainly has applications in the financial domain—most notably as a decentralized database for storing transactions of cryptographic tokens (cryptocurrencies). The potential use of blockchain technology for managing security issues in robot swarms was first outlined by Castelló Ferrer [4]. The author describes blockchain technology as the "key to serious progress in the field of swarm robotics" (p. 10). Several possible use cases, including secure communication, distributed decision making, and innovative business models are discussed in his paper. However, our paper is the first to provide an actual proof-of-concept of using blockchain technology for the coordination of robot swarms—including a description, implementation, and evaluation of the approach.

Related Work

- Summarize current works
- None is like us! (Novelty)
- Emphasize our novelty at last

Robot swarms are often assumed to be fault-tolerant *by design* [15], therefore, the explicit detection of faults was initially given little attention in swarm robotics research. Early security research focused on fault detection in the presence of *defective* robots [5, 6]. More recently, the explicit modeling of *malicious* robots has attracted more attention: [22] gives an overview of robot swarms' robustness to different attacker strategies in a cooperative navigation experiment and [28] presents a reputation management system that assigns a dynamic trust level to robots to identify malicious entities. In [10], a lightweight algorithm for detecting Sybil attacks via physical properties of wireless signals is developed. Connectivity requirements for achieving consensus in the presence of malicious robots are studied in [11]. [21] presents a resilient consensus protocol for dynamic agents whose network topology changes over time. However, so far, no secure general frameworks exist to cope with security issues in a fully decentralized way. Currently, blockchain technology mainly has applications in the financial domain—most notably as a decentralized database for storing transactions of cryptographic tokens (cryptocurrencies). The potential use of blockchain technology for managing security issues in robot swarms was first outlined by Castelló Ferrer [4]. The author describes blockchain technology as the "key to serious progress in the field of swarm robotics" (p. 10). Several possible use cases, including secure communication, distributed decision making, and innovative business models are discussed in his paper. However, our paper is the first to provide an actual proof-of-concept of using blockchain technology for the coordination of robot swarms—including a description, implementation, and evaluation of the approach.

2 RELATED WORK

Many studies (e.g., [2, 12, 13, 16, 19, 20, 25]) have addressed collective decision-making in robot swarms, a key task for the advancement of swarm robotics [7]. Collective decision-making tasks can be divided into two sub-classes: task allocation and consensus achievement [1]. In task allocation, the goal of the swarm is to maximize the overall swarm performance by assigning the members to different tasks. In consensus achievement, the goal of the swarm is to agree upon the best among a set of alternatives. The scenario that we use in this paper is based on the work of Valentini et al. [23], who describe and evaluate a collective decision scenario where robots have to reach consensus on the most frequent tile color in an environment in which the floor is covered with black and white tiles.

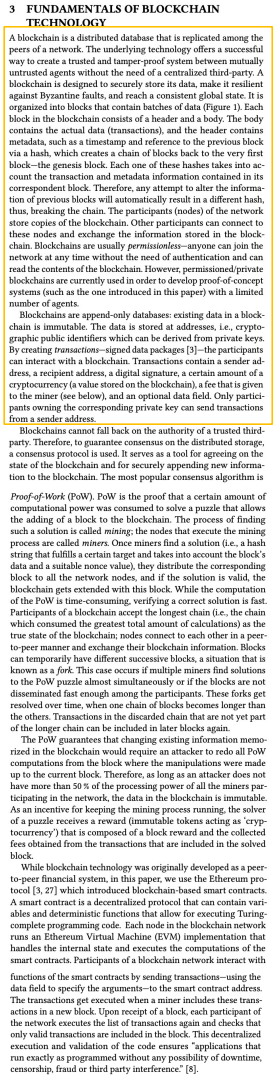
Robot swarms are often assumed to be fault-tolerant *by design* [15], therefore, the explicit detection of faults was initially given little attention in swarm robotics research. Early security research focused on fault detection in the presence of *defective* robots [5, 6]. More recently, the explicit modeling of *malicious* robots has attracted more attention: [22] gives an overview of robot swarms' robustness to different attacker strategies in a cooperative navigation experiment and [28] presents a reputation management system that assigns a dynamic trust level to robots to identify malicious entities. In [10], a lightweight algorithm for detecting Sybil attacks via physical properties of wireless signals is developed. Connectivity requirements for achieving consensus in the presence of malicious robots are studied in [11]. [21] presents a resilient consensus protocol for dynamic agents whose network topology changes over time. However, so far, no secure general frameworks exist to cope with security issues in a fully decentralized way. Currently, blockchain technology mainly has applications in the financial domain—most notably as a decentralized database for storing transactions of cryptographic tokens (cryptocurrencies). The potential use of blockchain technology for managing security issues in robot swarms was first outlined by Castelló Ferrer [4]. The author describes blockchain technology as the "key to serious progress in the field of swarm robotics" (p. 10). Several possible use cases, including secure communication, distributed decision making, and innovative business models are discussed in his paper. However, our paper is the first to provide an actual proof-of-concept of using blockchain technology for the coordination of robot swarms—including a description, implementation, and evaluation of the approach.

Fundamentals of Blockchain Technology

A blockchain is a distributed database that is replicated among the peers of a network. The underlying technology offers a successful way to create a trusted and tamper-proof system between mutually untrusted agents without the need of a centralized third-party. A blockchain is designed to securely store its data, make it resilient against Byzantine faults, and reach a consistent global state. It is organized into blocks that contain batches of data (Figure 1). Each block in the blockchain consists of a header and a body. The body contains the actual data (transactions), and the header contains metadata, such as a timestamp and reference to the previous block via a hash, which creates a chain of blocks back to the very first block—the genesis block. Each one of these hashes takes into account the transaction and metadata information contained in its correspondent block. Therefore, any attempt to alter the information of previous blocks will automatically result in a different hash, thus, breaking the chain. The participants (nodes) of the network store copies of the blockchain. Other participants can connect to these nodes and exchange the information stored in the blockchain. Blockchains are usually *permissionless*—anyone can join the network at any time without the need of authentication and can read the contents of the blockchain. However, permissioned/private blockchains are currently used in order to develop proof-of-concept systems (such as the one introduced in this paper) with a limited number of agents.

- Preliminary

Blockchains are append-only databases: existing data in a blockchain is immutable. The data is stored at addresses, i.e., cryptographic public identifiers which can be derived from private keys. By creating *transactions*—signed data packages [3]—the participants can interact with a blockchain. Transactions contain a sender address, a recipient address, a digital signature, a certain amount of a cryptocurrency (a value stored on the blockchain), a fee that is given to the miner (see below), and an optional data field. Only participants owning the corresponding private key can send transactions from a sender address.



3 FUNDAMENTALS OF BLOCKCHAIN TECHNOLOGY

A blockchain is a distributed database that is replicated among the peers of a network. The underlying technology offers a successful way to create a trusted and tamper-proof system between mutually untrusted agents without the need of a centralized third-party. A blockchain is designed to securely store its data, make it resilient against Byzantine faults, and reach a consistent global state. It is organized into blocks that contain batches of data (Figure 1). Each block in the blockchain consists of a header and a body. The body contains the actual data (transactions), and the header contains metadata, such as a timestamp and reference to the previous block via a hash, which creates a chain of blocks back to the very first block—the genesis block. Each one of these hashes takes into account the transaction and metadata information contained in its correspondent block. Therefore, any attempt to alter the information of previous blocks will automatically result in a different hash, thus, breaking the chain. The participants (nodes) of the network store copies of the blockchain. Other participants can connect to these nodes and exchange the information stored in the blockchain. Blockchains are usually *permissionless*—anyone can join the network at any time without the need of authentication and can read the contents of the blockchain. However, permissioned/private blockchains are currently used in order to develop proof-of-concept systems (such as the one introduced in this paper) with a limited number of agents.

Blockchains are append-only databases: existing data in a blockchain is immutable. The data is stored at addresses, i.e., cryptographic public identifiers which can be derived from private keys. By creating *transactions*—signed data packages [3]—the participants can interact with a blockchain. Transactions contain a sender address, a recipient address, a digital signature, a certain amount of a cryptocurrency (a value stored on the blockchain), a fee that is given to the miner (see below), and an optional data field. Only participants owning the corresponding private key can send transactions from a sender address.

Blockchains cannot fall back on the authority of a trusted third-party. Therefore, to guarantee consensus on the distributed storage, a consensus protocol is used. It serves as a tool for agreeing on the state of the blockchain and for securely appending new information to the blockchain. The most popular consensus algorithm is

Proof-of-Work (PoW). PoW is the proof that a certain amount of computational power was consumed to solve a puzzle that allows the adding of a block to the blockchain. The process of finding such a solution is called *mining*; the nodes that execute the mining process are called *miners*. Once miners find a solution (i.e., a hash string that fulfills a certain target and takes into account the block's data and a suitable nonce value), they distribute the corresponding block to all the network nodes, and if the solution is valid, the blockchain gets extended with this block. While the computation of the PoW is time-consuming, verifying a correct solution is fast. Participants of a blockchain accept the longest chain (i.e., the chain which consumed the greatest total amount of calculations) as the true state of the blockchain; nodes connect to each other in a peer-to-peer manner and exchange their blockchain information. Blocks can temporarily have different successive blocks, a situation that is known as a *fork*. This case occurs if multiple miners find solutions to the PoW puzzle almost simultaneously or if the blocks are not disseminated fast enough among the participants. These forks get resolved over time, when one chain of blocks becomes longer than the others. Transactions in the discarded chain that are not yet part of the longer chain can be included in later blocks again.

The PoW guarantees that changing existing information memorized in the blockchain would require an attacker to redo all PoW computations from the block where the manipulations were made up to the current block. Therefore, as long as an attacker does not have more than 50 % of the processing power of all the miners participating in the network, the data in the blockchain is immutable. As an incentive for keeping the mining process running, the solver of a puzzle receives a reward (immutable tokens acting as 'cryptocurrency') that is composed of a block reward and the collected fees obtained from the transactions that are included in the solved block.

While blockchain technology was originally developed as a peer-to-peer financial system, in this paper, we use the Ethereum protocol [3, 27] which introduced blockchain-based smart contracts. A smart contract is a decentralized protocol that can contain variables and deterministic functions that allow for executing Turing-complete programming code. Each node in the blockchain network runs an Ethereum Virtual Machine (EVM) implementation that handles the internal state and executes the computations of the smart contracts. Participants of a blockchain network interact with

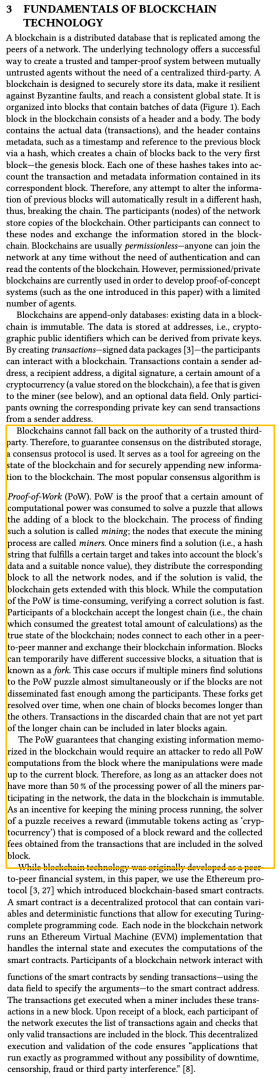
functions of the smart contracts by sending transactions—using the data field to specify the arguments—to the smart contract address. The transactions get executed when a miner includes these transactions in a new block. Upon receipt of a block, each participant of the network executes the list of transactions again and checks that only valid transactions are included in the block. This decentralized execution and validation of the code ensures "applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third party interference." [8].

Fundamentals of Blockchain Technology

- Preliminary

Blockchains cannot fall back on the authority of a trusted third-party. Therefore, to guarantee consensus on the distributed storage, a consensus protocol is used. It serves as a tool for agreeing on the state of the blockchain and for securely appending new information to the blockchain. The most popular consensus algorithm is *Proof-of-Work* (PoW). PoW is the proof that a certain amount of computational power was consumed to solve a puzzle that allows the adding of a block to the blockchain. The process of finding such a solution is called *mining*; the nodes that execute the mining process are called *miners*. Once miners find a solution (i.e., a hash string that fulfills a certain target and takes into account the block's data and a suitable nonce value), they distribute the corresponding block to all the network nodes, and if the solution is valid, the blockchain gets extended with this block. While the computation of the PoW is time-consuming, verifying a correct solution is fast. Participants of a blockchain accept the longest chain (i.e., the chain which consumed the greatest total amount of calculations) as the true state of the blockchain; nodes connect to each other in a peer-to-peer manner and exchange their blockchain information. Blocks can temporarily have different successive blocks, a situation that is known as a *fork*. This case occurs if multiple miners find solutions to the PoW puzzle almost simultaneously or if the blocks are not disseminated fast enough among the participants. These forks get resolved over time, when one chain of blocks becomes longer than the others. Transactions in the discarded chain that are not yet part of the longer chain can be included in later blocks again.

The PoW guarantees that changing existing information memorized in the blockchain would require an attacker to redo all PoW computations from the block where the manipulations were made up to the current block. Therefore, as long as an attacker does not have more than 50 % of the processing power of all the miners participating in the network, the data in the blockchain is immutable. As an incentive for keeping the mining process running, the solver of a puzzle receives a reward (immutable tokens acting as 'cryptocurrency') that is composed of a block reward and the collected fees obtained from the transactions that are included in the solved block.



3 FUNDAMENTALS OF BLOCKCHAIN TECHNOLOGY

A blockchain is a distributed database that is replicated among the peers of a network. The underlying technology offers a successful way to create a trusted and tamper-proof system between mutually untrusted agents without the need of a centralized third-party. A blockchain is designed to securely store its data, make it resilient against Byzantine faults, and reach a consistent global state. It is organized into blocks that contain batches of data (Figure 1). Each block in the blockchain consists of a header and a body. The body contains the actual data (transactions), and the header contains metadata, such as a timestamp and reference to the previous block via a hash, which creates a chain of blocks back to the very first block—the genesis block. Each one of these hashes takes into account the transaction and metadata information contained in its correspondent block. Therefore, any attempt to alter the information of previous blocks will automatically result in a different hash, thus, breaking the chain. The participants (nodes) of the network store copies of the blockchain. Other participants can connect to these nodes and exchange the information stored in the blockchain. Blockchains are usually *permissionless*—anyone can join the network at any time without the need of authentication and can read the contents of the blockchain. However, permissioned/private blockchains are currently used in order to develop proof-of-concept systems (such as the one introduced in this paper) with a limited number of agents.

Blockchains are append-only databases: existing data in a blockchain is immutable. The data is stored at addresses, i.e., cryptographic public identifiers which can be derived from private keys. By creating *transactions*—signed data packages [3]—the participants can interact with a blockchain. Transactions contain a sender address, a recipient address, a digital signature, a certain amount of a cryptocurrency (a value stored on the blockchain), a fee that is given to the miner (see below), and an optional data field. Only participants owning the corresponding private key can send transactions from a sender address.

Blockchains cannot fall back on the authority of a trusted third-party. Therefore, to guarantee consensus on the distributed storage, a consensus protocol is used. It serves as a tool for agreeing on the state of the blockchain and for securely appending new information to the blockchain. The most popular consensus algorithm is

Proof-of-Work (PoW). PoW is the proof that a certain amount of computational power was consumed to solve a puzzle that allows the adding of a block to the blockchain. The process of finding such a solution is called *mining*; the nodes that execute the mining process are called *miners*. Once miners find a solution (i.e., a hash string that fulfills a certain target and takes into account the block's data and a suitable nonce value), they distribute the corresponding block to all the network nodes, and if the solution is valid, the blockchain gets extended with this block. While the computation of the PoW is time-consuming, verifying a correct solution is fast. Participants of a blockchain accept the longest chain (i.e., the chain which consumed the greatest total amount of calculations) as the true state of the blockchain; nodes connect to each other in a peer-to-peer manner and exchange their blockchain information. Blocks can temporarily have different successive blocks, a situation that is known as a *fork*. This case occurs if multiple miners find solutions to the PoW puzzle almost simultaneously or if the blocks are not disseminated fast enough among the participants. These forks get resolved over time, when one chain of blocks becomes longer than the others. Transactions in the discarded chain that are not yet part of the longer chain can be included in later blocks again.

The PoW guarantees that changing existing information memorized in the blockchain would require an attacker to redo all PoW computations from the block where the manipulations were made up to the current block. Therefore, as long as an attacker does not have more than 50 % of the processing power of all the miners participating in the network, the data in the blockchain is immutable. As an incentive for keeping the mining process running, the solver of a puzzle receives a reward (immutable tokens acting as 'cryptocurrency') that is composed of a block reward and the collected fees obtained from the transactions that are included in the solved block.

While blockchain technology was originally developed as a peer-to-peer financial system, in this paper, we use the Ethereum protocol [3, 27] which introduced blockchain-based smart contracts. A smart contract is a decentralized protocol that can contain variables and deterministic functions that allow for executing Turing-complete programming code. Each node in the blockchain network runs an Ethereum Virtual Machine (EVM) implementation that handles the internal state and executes the computations of the smart contracts. Participants of a blockchain network interact with functions of the smart contracts by sending transactions—using the data field to specify the arguments—to the smart contract address. The transactions get executed when a miner includes these transactions in a new block. Upon receipt of a block, each participant of the network executes the list of transactions again and checks that only valid transactions are included in the block. This decentralized execution and validation of the code ensures "applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third party interference." [8].

Fundamentals of Blockchain Technology

- Preliminary

While blockchain technology was originally developed as a peer-to-peer financial system, in this paper, we use the Ethereum protocol [3, 27] which introduced blockchain-based smart contracts. A smart contract is a decentralized protocol that can contain variables and deterministic functions that allow for executing Turing-complete programming code. Each node in the blockchain network runs an Ethereum Virtual Machine (EVM) implementation that handles the internal state and executes the computations of the smart contracts. Participants of a blockchain network interact with functions of the smart contracts by sending transactions—using the data field to specify the arguments—to the smart contract address. The transactions get executed when a miner includes these transactions in a new block. Upon receipt of a block, each participant of the network executes the list of transactions again and checks that only valid transactions are included in the block. This decentralized execution and validation of the code ensures "applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third party interference." [8].

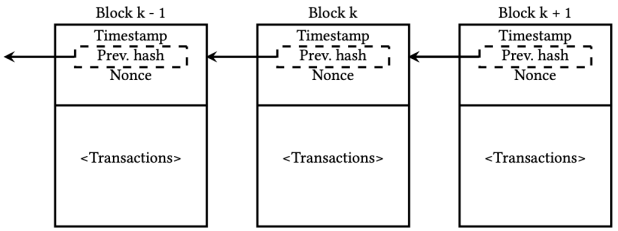


Figure 1: The blockchain—a distributed database—is organized into blocks. Each block consists of two parts, the header and the body. The header contains metadata for the block—most importantly the hash of the previous block which creates a unique chain of blocks. The body contains the actual data: the transactions.

3 FUNDAMENTALS OF BLOCKCHAIN TECHNOLOGY

A blockchain is a distributed database that is replicated among the peers of a network. The underlying technology offers a successful way to create a trusted and tamper-proof system between mutually untrusted agents without the need of a centralized third-party. A blockchain is designed to securely store its data, make it resilient against Byzantine faults, and reach a consistent global state. It is organized into blocks that contain batches of data (Figure 1). Each block in the blockchain consists of a header and a body. The body contains the actual data (transactions), and the header contains metadata, such as a timestamp and reference to the previous block via a hash, which creates a chain of blocks back to the very first block—the genesis block. Each one of these hashes takes into account the transaction and metadata information contained in its correspondent block. Therefore, any attempt to alter the information of previous blocks will automatically result in a different hash, thus, breaking the chain. The participants (nodes) of the network store copies of the blockchain. Other participants can connect to these nodes and exchange the information stored in the blockchain. Blockchains are usually *permissionless*—anyone can join the network at any time without the need of authentication and can read the contents of the blockchain. However, permissioned/private blockchains are currently used in order to develop proof-of-concept systems (such as the one introduced in this paper) with a limited number of agents.

Blockchains are append-only databases: existing data in a blockchain is immutable. The data is stored at addresses, i.e., cryptographic public identifiers which can be derived from private keys. By creating *transactions*—signed data packages [3]—the participants can interact with a blockchain. Transactions contain a sender address, a recipient address, a digital signature, a certain amount of a cryptocurrency (a value stored on the blockchain), a fee that is given to the miner (see below), and an optional data field. Only participants owning the corresponding private key can send transactions from a sender address.

Blockchains cannot fall back on the authority of a trusted third-party. Therefore, to guarantee consensus on the distributed storage, a consensus protocol is used. It serves as a tool for agreeing on the state of the blockchain and for securely appending new information to the blockchain. The most popular consensus algorithm is

Proof-of-Work (PoW). PoW is the proof that a certain amount of computational power was consumed to solve a puzzle that allows the adding of a block to the blockchain. The process of finding such a solution is called *mining*; the nodes that execute the mining process are called *miners*. Once miners find a solution (i.e., a hash string that fulfills a certain target and takes into account the block's data and a suitable nonce value), they distribute the corresponding block to all the network nodes, and if the solution is valid, the blockchain gets extended with this block. While the computation of the PoW is time-consuming, verifying a correct solution is fast. Participants of a blockchain accept the longest chain (i.e., the chain which consumed the greatest total amount of calculations) as the true state of the blockchain; nodes connect to each other in a peer-to-peer manner and exchange their blockchain information. Blocks can temporarily have different successive blocks, a situation that is known as a *fork*. This case occurs if multiple miners find solutions to the PoW puzzle almost simultaneously or if the blocks are not disseminated fast enough among the participants. These forks get resolved over time, when one chain of blocks becomes longer than the others. Transactions in the discarded chain that are not yet part of the longer chain can be included in later blocks again.

The PoW guarantees that changing existing information memorized in the blockchain would require an attacker to redo all PoW computations from the block where the manipulations were made up to the current block. Therefore, as long as an attacker does not have more than 50 % of the processing power of all the miners participating in the network, the data in the blockchain is immutable. As an incentive for keeping the mining process running, the solver of a puzzle receives a reward (immutable tokens acting as 'cryptocurrency') that is composed of a block reward and the collected fees obtained from the transactions that are included in the solved block.

While blockchain technology was originally developed as a peer-to-peer financial system, in this paper, we use the Ethereum protocol [3, 27] which introduced blockchain-based smart contracts. A smart contract is a decentralized protocol that can contain variables and deterministic functions that allow for executing Turing-complete programming code. Each node in the blockchain network runs an Ethereum Virtual Machine (EVM) implementation that handles the internal state and executes the computations of the smart contracts. Participants of a blockchain network interact with functions of the smart contracts by sending transactions—using the data field to specify the arguments—to the smart contract address. The transactions get executed when a miner includes these transactions in a new block. Upon receipt of a block, each participant of the network executes the list of transactions again and checks that only valid transactions are included in the block. This decentralized execution and validation of the code ensures "applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third party interference." [8].

Methods

4 METHODS

In this section, we present Valentini et al.'s work [23] (*classical approach*) and our approach that uses blockchain-based smart contracts (*blockchain approach*). We use the subscripts 'cl' and 'bc' to denote the classical and blockchain variants of elements in our algorithms, respectively (e.g., $\mathrm{DC}_{\mathrm{bc}}$ for the Direct Comparison (DC) strategy of the blockchain approach—defined below).

- Define the notation
- Explain the goal of the task
- Quantify the level of difficulty
- Define what is a successfully run

4.1 Experimental setup

In both the classical and blockchain approaches, the goal of the robot swarm is to make a collective decision and to reach consensus on the most frequent tile color (in all our experiments the white color) of a black/white grid (Figure 2)[1]. Each robot has a current opinion about which color is the most frequent, and via dissemination/decision-making strategies, they influence their peers.

The robots move in a square environment of 2×2 m^2 that is bounded by four walls. The grid is composed of $|B|$ black tiles and $|W|$ white tiles, with $|B| + |W| = 400$. Each tile is 0.1×0.1 m^2. The difficulty of the task (ρ_b^*) can be varied by modifying the ratio between the percentage $\rho_b = \frac{|B|}{|B|+|W|}$ of black tiles and $\rho_w = \frac{|W|}{|B|+|W|}$ of white tiles: $\rho_b^* = \frac{\rho_b}{\rho_w}$. In a relatively simple task, the difference between the percentage of white and black tiles is large (e.g., $\rho_b = 34\,\%$, $\rho_w = 66\,\% \rightarrow \rho_b^* \approx 0.52$), while in a relatively difficult task, the difference is small (e.g., $\rho_b = 48\,\%$, $\rho_w = 52\,\% \rightarrow \rho_b^* \approx 0.92$).

At the end of a successful run, all robots will have the same opinion corresponding to the most frequent color.

Methods

4.2 Simulation environment

The simulations were executed in discrete time steps (ticks)—with 10 ticks per second using the ARGoS robot swarm simulator [18] (version 3.0.0-beta48) and the ARGoS-Epuck [9] plugin with $N = 20$ e-puck robots on a computer cluster. For each experimental run, two nodes were used on the cluster with 16 cores each. Each core has a clock rate of 2.0 GHz and 1 GB of RAM. ARGoS was executed using $N = 20$ threads, and, for the blockchain approach, one *geth* process (an interface for running a full Ethereum node) was started for each robot using a single thread. The simulated robots were programmed to use the IPC socket of *geth*.

Robots can only communicate with each other if there is no other robot hindering the communication and if the robots' distance to each other is smaller than $d_n = 50\,\text{cm}$. We used this design constraint in order to mimic the capabilities of IR/Bluetooth/RF transmission, so that we can easily test the algorithm using real robots in future research.

- Values for some hyper-parameters of the simulator
- Explain some design choice

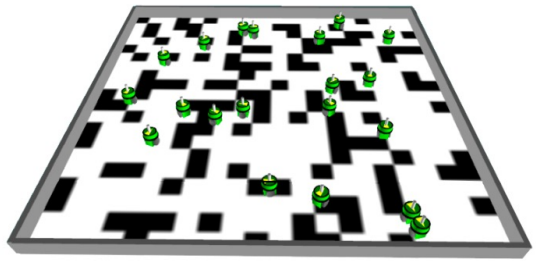
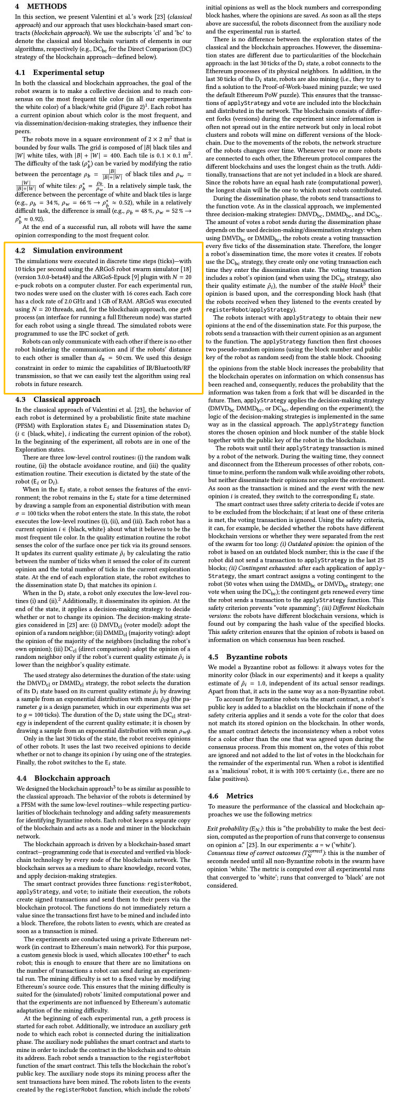
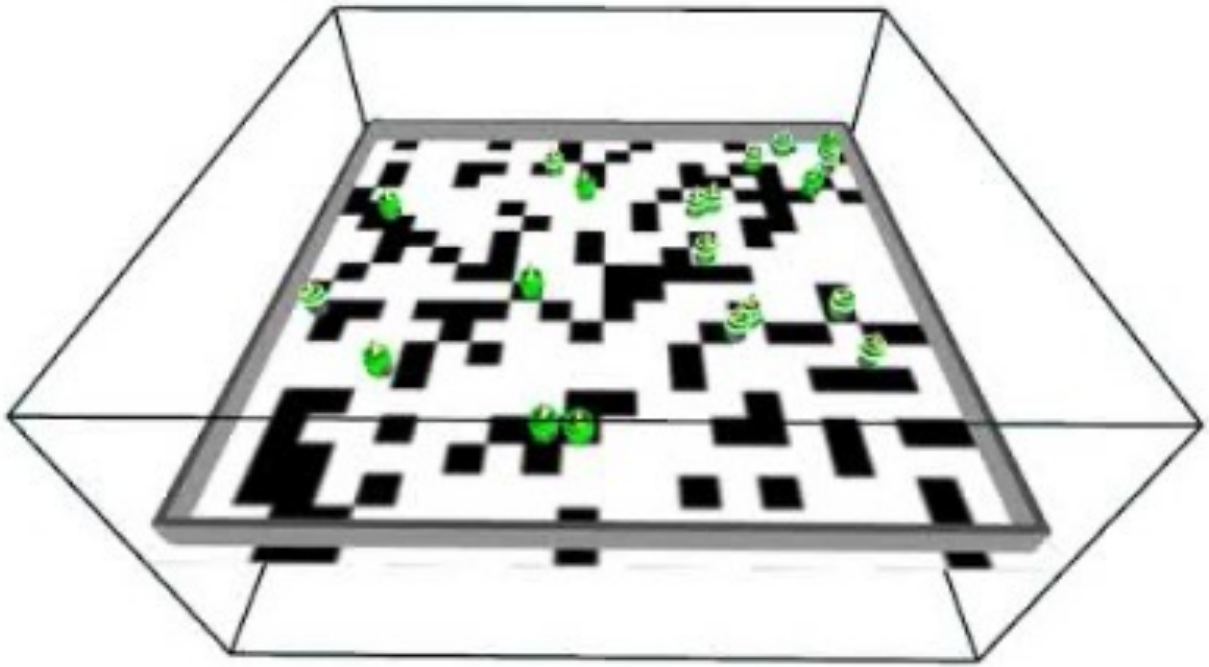


Figure 2: The robots' task is to determine the most frequent color in an environment whose floor is covered with black and white tiles. This collective-decision experiment was conducted using the ARGoS robot swarm simulator with the plugin for e-puck robots.





Methods

4.3 Classical approach

In the classical approach of Valentini et al. [23], the behavior of each robot is determined by a probabilistic finite state machine (PFSM) with Exploration states E_i and Dissemination states D_i ($i \in \{\text{black}, \text{white}\}$, i indicating the current opinion of the robot). In the beginning of the experiment, all robots are in one of the Exploration states.

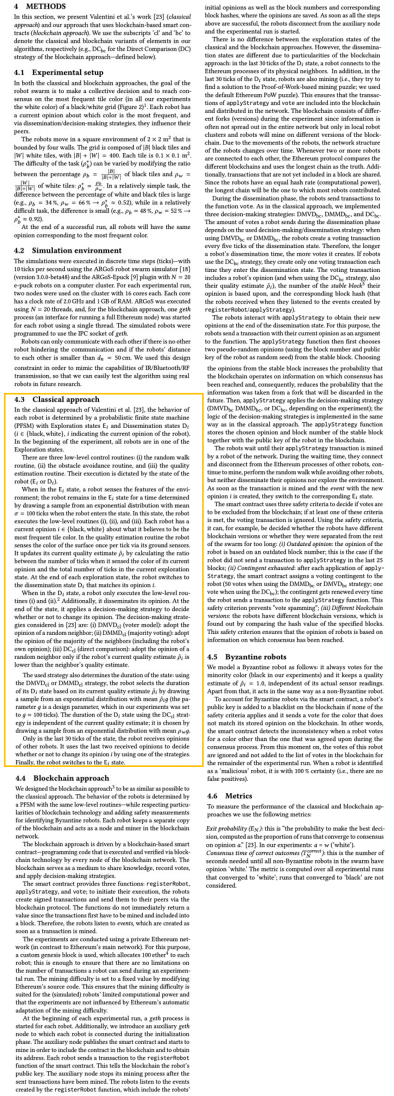
There are three low-level control routines: (i) the random walk routine, (ii) the obstacle avoidance routine, and (iii) the quality estimation routine. Their execution is dictated by the state of the robot (E_i or D_i).

When in the E_i state, a robot senses the features of the environment; the robot remains in the E_i state for a time determined by drawing a sample from an exponential distribution with mean $\sigma = 100$ ticks when the robot enters the state. In this state, the robot executes the low-level routines (i), (ii), and (iii). Each robot has a current opinion $i \in \{\text{black}, \text{white}\}$ about what it believes to be the most frequent tile color. In the quality estimation routine the robot senses the color of the surface once per tick via its ground sensors. It updates its current quality estimate $\hat{\rho}_i$ by calculating the ratio between the number of ticks when it sensed the color of its current opinion and the total number of ticks in the current exploration state. At the end of each exploration state, the robot switches to the dissemination state D_i that matches its opinion i.

When in the D_i state, a robot only executes the low-level routines (i) and (ii).[2] Additionally, it disseminates its opinion. At the end of the state, it applies a decision-making strategy to decide whether or not to change its opinion. The decision-making strategies considered in [23] are: (i) DMVD_{cl} (voter model): adopt the opinion of a random neighbor; (ii) DMMD_{cl} (majority voting): adopt the opinion of the majority of the neighbors (including the robot's own opinion); (iii) DC_{cl} (direct comparison): adopt the opinion of a random neighbor only if the robot's current quality estimate $\hat{\rho}_i$ is lower than the neighbor's quality estimate.

The used strategy also determines the duration of the state: using the DMVD_{cl} or DMMD_{cl} strategy, the robot selects the duration of its D_i state based on its current quality estimate $\hat{\rho}_i$ by drawing a sample from an exponential distribution with mean $\hat{\rho}_i g$ (the parameter g is a design parameter, which in our experiments was set to $g = 100$ ticks). The duration of the D_i state using the DC_{cl} strategy is independent of the current quality estimate; it is chosen by drawing a sample from an exponential distribution with mean $\rho_w g$.

Only in the last 30 ticks of the state, the robot receives opinions of other robots. It uses the last two received opinions to decide whether or not to change its opinion i by using one of the strategies. Finally, the robot switches to the E_i state.



Methods

4.4 Blockchain approach

We designed the blockchain approach[3] to be as similar as possible to the classical approach. The behavior of the robots is determined by a PFSM with the same low-level routines—while respecting particularities of blockchain technology and adding safety measurements for identifying Byzantine robots. Each robot keeps a separate copy of the blockchain and acts as a node and miner in the blockchain network.

The blockchain approach is driven by a blockchain-based smart contract—programming code that is executed and verified via blockchain technology by every node of the blockchain network. The blockchain serves as a medium to share knowledge, record votes, and apply decision-making strategies.

The smart contract provides three functions: `registerRobot`, `applyStrategy`, and `vote`; to initiate their execution, the robots create signed transactions and send them to their peers via the blockchain protocol. The functions do not immediately return a value since the transactions first have to be mined and included into a block. Therefore, the robots listen to *events*, which are created as soon as a transaction is mined.

The experiments are conducted using a private Ethereum network (in contrast to Ethereum's main network). For this purpose, a custom genesis block is used, which allocates 100 ether[4] to each robot; this is enough to ensure that there are no limitations on the number of transactions a robot can send during an experimental run. The mining difficulty is set to a fixed value by modifying Ethereum's source code. This ensures that the mining difficulty is suited for the (simulated) robots' limited computational power and that the experiments are not influenced by Ethereum's automatic adaptation of the mining difficulty.

At the beginning of each experimental run, a *geth* process is started for each robot. Additionally, we introduce an auxiliary *geth* node to which each robot is connected during the initialization phase. The auxiliary node publishes the smart contract and starts to mine in order to include the contract in the blockchain and to obtain its address. Each robot sends a transaction to the `registerRobot` function of the smart contract. This tells the blockchain the robot's public key. The auxiliary node stops its mining process after the sent transactions have been mined. The robots listen to the events created by the `registerRobot` function, which include the robots' initial opinions as well as the block numbers and corresponding block hashes, where the opinions are saved. As soon as all the steps above are successful, the robots disconnect from the auxiliary node and the experimental run is started.

There is no difference between the exploration states of the classical and the blockchain approaches. However, the dissemination states are different due to particularities of the blockchain approach: in the last 30 ticks of the D_i state, a robot connects to the Ethereum processes of its physical neighbors. In addition, in the last 30 ticks of the D_i state, robots are also mining (i.e., they try to find a solution to the Proof-of-Work-based mining puzzle; we used the default Ethereum PoW puzzle). This ensures that the transactions of `applyStrategy` and `vote` are included into the blockchain and distributed in the network. The blockchain consists of different forks (versions) during the experiment since information is often not spread out in the entire network but only in local robot clusters and robots will mine on different versions of the blockchain. Due to the movements of the robots, the network structure of the robots changes over time. Whenever two or more robots are connected to each other, the Ethereum protocol compares the different blockchains and uses the longest chain as the truth. Additionally, transactions that are not yet included in a block are shared. Since the robots have an equal hash rate (computational power), the longest chain will be the one to which most robots contributed.

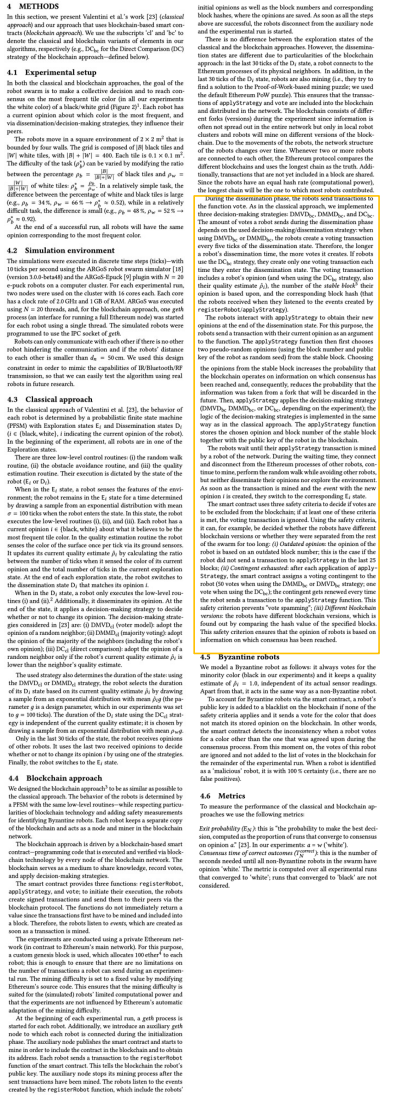
Methods

During the dissemination phase, the robots send transactions to the function `vote`. As in the classical approach, we implemented three decision-making strategies: DMVD_{bc}, DMMD_{bc}, and DC_{bc}. The amount of votes a robot sends during the dissemination phase depends on the used decision-making/dissemination strategy: when using DMVD_{bc} or DMMD_{bc}, the robots create a voting transaction every five ticks of the dissemination state. Therefore, the longer a robot's dissemination time, the more votes it creates. If robots use the DC_{bc} strategy, they create only one voting transaction each time they enter the dissemination state. The voting transaction includes a robot's opinion (and when using the DC_{bc} strategy, also their quality estimate $\hat{\rho}_i$), the number of the *stable block*[5] their opinion is based upon, and the corresponding block hash (that the robots received when they listened to the events created by `registerRobot/applyStrategy`).

The robots interact with `applyStrategy` to obtain their new opinions at the end of the dissemination state. For this purpose, the robots send a transaction with their current opinion as an argument to the function. The `applyStrategy` function then first chooses two pseudo-random opinions (using the block number and public key of the robot as random seed) from the stable block. Choosing the opinions from the stable block increases the probability that the blockchain operates on information on which consensus has been reached and, consequently, reduces the probability that the information was taken from a fork that will be discarded in the future. Then, `applyStrategy` applies the decision-making strategy (DMVD_{bc} DMMD_{bc}, or DC_{bc}, depending on the experiment); the logic of the decision-making strategies is implemented in the same way as in the classical approach. The `applyStrategy` function stores the chosen opinion and block number of the stable block together with the public key of the robot in the blockchain.

The robots wait until their `applyStrategy` transaction is mined by a robot of the network. During the waiting time, they connect and disconnect from the Ethereum processes of other robots, continue to mine, perform the random walk while avoiding other robots, but neither disseminate their opinions nor explore the environment. As soon as the transaction is mined and the *event* with the new opinion i is created, they switch to the corresponding E_i state.

The smart contract uses three safety criteria to decide if votes are to be excluded from the blockchain; if at least one of these criteria is met, the voting transaction is ignored. Using the safety criteria, it can, for example, be decided whether the robots have different blockchain versions or whether they were separated from the rest of the swarm for too long: *(i) Outdated opinion*: the opinion of the robot is based on an outdated block number; this is the case if the robot did not send a transaction to `applyStrategy` in the last 25 blocks; *(ii) Contingent exhausted*: after each application of `apply-Strategy`, the smart contract assigns a voting contingent to the robot (50 votes when using the DMMD_{bc} or DMVD_{bc} strategy; one vote when using the DC_{bc}); the contingent gets renewed every time the robot sends a transaction to the `applyStrategy` function. This safety criterion prevents "vote spamming"; *(iii) Different blockchain versions*: the robots have different blockchain versions, which is found out by comparing the hash value of the specified blocks. This safety criterion ensures that the opinion of robots is based on information on which consensus has been reached.

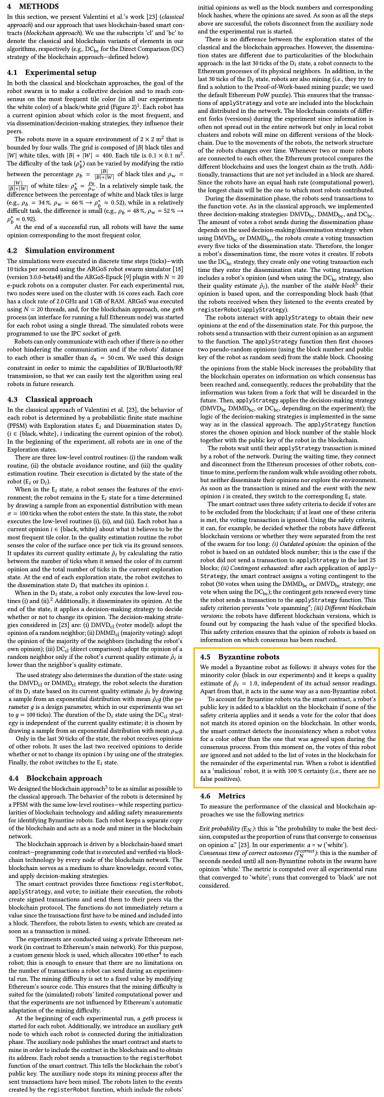


Methods

4.5 Byzantine robots

We model a Byzantine robot as follows: it always votes for the minority color (black in our experiments) and it keeps a quality estimate of $\hat{\rho}_i = 1.0$, independent of its actual sensor readings. Apart from that, it acts in the same way as a non-Byzantine robot.

To account for Byzantine robots via the smart contract, a robot's public key is added to a blacklist on the blockchain if none of the safety criteria applies and it sends a vote for the color that does not match its stored opinion on the blockchain. In other words, the smart contract detects the inconsistency when a robot votes for a color other than the one that was agreed upon during the consensus process. From this moment on, the votes of this robot are ignored and not added to the list of votes in the blockchain for the remainder of the experimental run. When a robot is identified as a 'malicious' robot, it is with 100 % certainty (i.e., there are no false positives).



4 METHODS

In this section, we present Valentini et al.'s work [25] (*classical approach*) and our approach that uses blockchain-based smart contracts (*blockchain approach*). We use the subscripts 'cl' and 'bc' to denote the classical and blockchain variants of elements in our algorithms, respectively (e.g., DC_{bc} for the Direct Comparison (DC) strategy of the blockchain approach—defined below).

4.1 Experimental setup

In both the classical and blockchain approaches, the goal of the robot swarm is to make a collective decision and to reach consensus on the most frequent tile color (in all our experiments the white color) of a black/white grid (Figure 2)[2]. Each robot has a current opinion about which color is the most frequent, and via dissemination/decision-making strategies, they influence their peers.

The robots move in a square environment of 2×2 m^2 that is bounded by four walls. The grid is composed of $|B|$ black tiles and $|W|$ white tiles, with $|B| + |W| = 400$. Each tile is 0.1×0.1 m^2. The difficulty of the task (ρ_d^*) can be varied by modifying the ratio between the percentage $\rho_B = \frac{|B|}{|B|+|W|}$ of black tiles and $\rho_w = \frac{|W|}{|B|+|W|}$ of white tiles: $\rho_d^* = \frac{\rho_w}{\rho_B}$. In a relatively simple task, the difference between the percentage of white and black tiles is large (e.g., $\rho_B = 34\,\%$, $\rho_w = 66\,\% \rightarrow \rho_d^* \approx 0.52$), while in a relatively difficult task, the difference is small (e.g., $\rho_B = 48\,\%$, $\rho_w = 52\,\% \rightarrow \rho_d^* \approx 0.92$).

At the end of a successful run, all robots will have the same opinion corresponding to the most frequent color.

4.2 Simulation environment

The simulations were executed in discrete time steps (ticks)—with 10 ticks per second using the ARGoS robot swarm simulator [18] (version 3.0.0-beta48) and the ARGoS-Epuck [9] plugin with $N = 20$ e-puck robots on a computer cluster. For each experimental run, two nodes were used on the cluster with 16 cores each. Each core has a clock rate of 2.0 GHz and 1 GB of RAM. ARGoS was executed using $N = 20$ threads, and, for the blockchain approach, one *geth* process (an interface for running a full Ethereum node) was started for each robot using a single thread. The simulated robots were programmed to use the IPC socket of *geth*.

Robots can only communicate with each other if there is no other robot hindering the communication and if the robots' distance to each other is smaller than $d_n = 50$ cm. We used this design constraint in order to mimic the capabilities of IR/Bluetooth/RF transmission, so that we can easily test the algorithm using real robots in future research.

4.3 Classical approach

In the classical approach of Valentini et al. [25], the behavior of each robot is determined by a probabilistic finite state machine (PFSM) with Exploration states E_i and Dissemination states D_i ($i \in$ {black, white}, i indicating the current opinion of the robot). In the beginning of the experiment, all robots are in one of the Exploration states.

There are three low-level control routines: (i) the random walk routine, (ii) the obstacle avoidance routine, and (iii) the quality estimation routine. Their execution is dictated by the state of the robot (E_i or D_i).

When in the E_i state, a robot senses the features of the environment; the robot remains in the E_i state for a time determined by drawing a sample from an exponential distribution with mean $\sigma = 100$ ticks when the robot enters the state. In this state, the robot executes the low-level routines (i), (ii), and (iii). Each robot has a current opinion $i \in$ {black, white} about what it believes to be the most frequent tile color. In the quality estimation routine the robot senses the color of the surface once per tick via its ground sensors. It updates its current quality estimate $\hat{\rho}_i$ by calculating the ratio between the number of ticks when it sensed the color of its current opinion and the total number of ticks in the current exploration state. At the end of each exploration state, the robot switches to the dissemination state D_i that matches its opinion i.

When in the D_i state, a robot only executes the low-level routines (i) and (ii)[3]. Additionally, it disseminates its opinion. At the end of the state, it applies a decision-making strategy to decide whether or not to change its opinion. The decision-making strategies considered in [25] are: (i) $DMVD_{cl}$ (voter model): adopt the opinion of a random neighbor; (ii) $DMMD_{cl}$ (majority voting): adopt the opinion of the majority of the neighbors (including the robot's own opinion); (iii) DC_{cl} (direct comparison): adopt the opinion of a random neighbor only if the robot's current quality estimate $\hat{\rho}_i$ is lower than the neighbor's quality estimate.

The used strategy also determines the duration of the state: using the $DMVD_{cl}$ or $DMMD_{cl}$ strategy, the robot selects the duration of its D_i state based on its current quality estimate $\hat{\rho}_i$ by drawing a sample from an exponential distribution with mean $\hat{\rho}_i g$ (the parameter g is a design parameter, which in our experiments was set to $g = 100$ ticks). The duration of the D_i state using the DC_{cl} strategy is independent of the current quality estimate; it is chosen by drawing a sample from an exponential distribution with mean $\rho_* g$.

Only in the last 30 ticks of the state, the robot receives opinions of other robots. It uses the last two received opinions to decide whether or not to change its opinion i by using one of the strategies. Finally, the robot switches to the E_i state.

4.4 Blockchain approach

We designed the blockchain approach[3] to be as similar as possible to the classical approach. The behavior of the robots is determined by a PFSM with the same low-level routines—while respecting particularities of blockchain technology and adding safety measurements for identifying Byzantine robots. Each robot keeps a separate copy of the blockchain and acts as a node and miner in the blockchain network.

The blockchain approach is driven by a blockchain-based smart contract—programming code that is executed and verified via blockchain technology by every node of the blockchain network. The blockchain serves as a medium to share knowledge, record votes, and apply decision-making strategies.

The smart contract provides three functions: `registerRobot`, `applyStrategy`, and `vote`, to initiate their execution, the robots create signed transactions and send them to their peers via the blockchain protocol. The functions do not immediately return a value since the transactions first have to be mined and included into a block. Therefore, the robots listen to *events*, which are created as soon as a transaction is mined.

The experiments are conducted using a private Ethereum network (in contrast to Ethereum's main network). For this purpose, a custom genesis block is used, which allocates 100 ether[4] to each robot; this is enough to ensure that there are no limitations on the number of transactions a robot can send during an experimental run. The mining difficulty is set to a fixed value by modifying Ethereum's source code. This ensures that the mining difficulty is suited for the (simulated) robots' limited computational power and that the experiments are not influenced by Ethereum's automatic adaptation of the mining difficulty.

At the beginning of each experimental run, a *geth* process is started for each robot. Additionally, we introduce an auxiliary *geth* node to which each robot is connected during the initialization phase. The auxiliary node publishes the smart contract and starts to mine in order to include the contract in the blockchain and to obtain its address. Each robot sends a transaction to the `registerRobot` function of the smart contract. This tells the blockchain the robot's public key. The auxiliary node stops its mining process after the sent transactions have been mined. The robots listen to the events created by the `registerRobot` function, which include the robots' initial opinions as well as the block numbers and corresponding block hashes, where the opinions are saved. As soon as all the steps above are successful, the robots disconnect from the auxiliary node and the experimental run is started.

There is no difference between the exploration states of the classical and the blockchain approaches. However, the dissemination states are different due to particularities of the blockchain approach: in the last 30 ticks of the D_i state, a robot connects to the Ethereum processes of its physical neighbors. In addition, in the last 30 ticks of the D_i state, robots are also mining (i.e., they try to find a solution to the Proof-of-Work-based mining puzzle; we used the default Ethereum PoW puzzle). This ensures that the transactions of `applyStrategy` and `vote` are included into the blockchain and distributed in the network. The blockchain consists of different forks (versions) during the experiment since information is often not spread out in the entire network but only in local robot clusters and robots will mine on different versions of the blockchain. Due to the movements of the robots, the network structure of the robots changes over time. Whenever two or more robots are connected to each other, the Ethereum protocol compares the different blockchains and uses the longest chain as the truth. Additionally, transactions that are not yet included in a block are shared. Since the robots have an equal hash rate (computational power), the longest chain will be the one to which most robots contributed.

During the dissemination phase, the robots send transactions to the function `vote`. As in the classical approach, we implemented three decision-making strategies: $DMVD_{bc}$, $DMMD_{bc}$, and DC_{bc}. The amount of votes a robot sends during the dissemination phase depends on the used decision-making/dissemination strategy: when using $DMVD_{bc}$ or $DMMD_{bc}$, the robots create a voting transaction every five ticks of the dissemination state. Therefore, the longer a robot's dissemination time, the more votes it creates. If robots use the DC_{bc} strategy, they create only one voting transaction each time they enter the dissemination state. The voting transaction includes a robot's opinion (and when using the DC_{bc} strategy, also their quality estimate $\hat{\rho}_i$), the number of the stable block[5] their opinion is based upon, and the corresponding block hash (that the robots received when they listened to the events created by `registerRobot`/`applyStrategy`).

The robots interact with `applyStrategy` to obtain their new opinions at the end of the dissemination state. For this purpose, the robots send a transaction with their current opinion as an argument to the function. The `applyStrategy` function then first chooses two pseudo-random opinions (using the block number and public key of the robot as random seed) from the stable block. Choosing the opinions from the stable block increases the probability that the blockchain operates on information on which consensus has been reached and, consequently, reduces the probability that the information was taken from a fork that will be discarded in the future. Then, `applyStrategy` applies the decision-making strategy ($DMVD_{bc}$, $DMMD_{bc}$, or DC_{bc}, depending on the experiment), the logic of the decision-making strategies is implemented in the same way as in the classical approach. The `applyStrategy` function stores the chosen opinion and block number of the stable block together with the public key of the robot in the blockchain.

The robots wait until their `applyStrategy` transaction is mined by a robot of the network. During the waiting time, they connect and disconnect from the Ethereum processes of other robots, continue to mine, perform the random walk while avoiding other robots, but neither disseminate their opinions nor explore the environment. As soon as the transaction is mined and the *event* with the new opinion i is created, they switch to the corresponding E_i state.

The smart contract uses three safety criteria to decide if votes are to be excluded from the blockchain; if at least one of these criteria is met, the voting transaction is ignored. Using the safety criteria, it can, for example, be decided whether the robots have different blockchain versions or whether they were separated from the rest of the swarm for too long: *(i) Outdated opinion*: the opinion of the robot is based on an outdated block number; this is the case if the robot did not send a transaction to `applyStrategy` in the last 25 blocks; *(ii) Contingent exhausted*: after each application of `applyStrategy`, the smart contract assigns a voting contingent to the robot (50 votes when using the $DMMD_{bc}$ or $DMVD_{bc}$ strategy; one vote when using the DC_{bc}); the contingent gets renewed every time the robot sends a transaction to the `applyStrategy` function. This safety criterion prevents "vote spamming"; *(iii) Different blockchain versions*: the robots have different blockchain versions, which is found out by comparing the hash value of the specified blocks. This safety criterion ensures that the opinion of robots is based on information on which consensus has been reached.

4.5 Byzantine robots

We model a Byzantine robot as follows: it always votes for the minority color (black in our experiments) and it keeps a quality estimate of $\hat{\rho}_i = 1.0$, independent of its actual sensor readings. Apart from that, it acts in the same way as a non-Byzantine robot.

To account for Byzantine robots via the smart contract, a robot's public key is added to a blacklist on the blockchain if none of the safety criteria applies and it sends a vote for the color that does not match its stored opinion on the blockchain. In other words, the smart contract detects the inconsistency when a robot votes for a color other than the one that was agreed upon during the consensus process. From this moment on, the votes of this robot are ignored and not added to the list of votes in the blockchain for the remainder of the experimental run. When a robot is identified as a 'malicious' robot, it is with 100 % certainty (i.e., there are no false positives).

4.6 Metrics

To measure the performance of the classical and blockchain approaches we use the following metrics:

Exit probability (E_{bc}): this is "the probability to make the best decision, computed as the proportion of runs that converge to consensus on opinion a^* [25]. In our experiments $a^* = w$ ('white').

Consensus time of correct outcomes ($T_c^{conv(a^*)}$): this is the number of seconds needed until all non-Byzantine robots in the swarm have opinion 'white.' The metric is computed over all experimental runs that converged to 'white'; runs that converged to 'black' are not considered.

Methods

4.6 Metrics

To measure the performance of the classical and blockchain approaches we use the following metrics:

Exit probability (E_N): this is "the probability to make the best decision, computed as the proportion of runs that converge to consensus on opinion a." [23]. In our experiments: $a = w$ ('white').

Consensus time of correct outcomes ($T_N^{correct}$): this is the number of seconds needed until all non-Byzantine robots in the swarm have opinion 'white.' The metric is computed over all experimental runs that converged to 'white'; runs that converged to 'black' are not considered.

4 METHODS

In this section, we present Valentini et al.'s work [23] (*classical approach*) and our approach that uses blockchain-based smart contracts (*blockchain approach*). We use the subscripts 'cl' and 'bc' to denote the classical and blockchain variants of elements in our algorithms, respectively (e.g., DC_{bc} for the Direct Comparison (DC) strategy of the blockchain approach—defined below).

4.1 Experimental setup

In both the classical and blockchain approaches, the goal of the robot swarm is to make a collective decision and to reach consensus on the most frequent tile color (in all our experiments the white color) of a black/white grid (Figure 2)[2]. Each robot has a current opinion about which color is the most frequent, and via dissemination/decision-making strategies, they influence their peers.

The robots move in a square environment of 2×2 m^2 that is bounded by four walls. The grid is composed of $|B|$ black tiles and $|W|$ white tiles, with $|B| + |W| = 400$. Each tile is 0.1×0.1 m^2. The difficulty of the task (ρ_N^*) can be varied by modifying the ratio between the percentage $\rho_B = \frac{|B|}{|B|+|W|}$ of black tiles and $\rho_w = \frac{|W|}{|B|+|W|}$ of white tiles: $\rho_N^* = \frac{\rho_w}{\rho_B}$. In a relatively simple task, the difference between the percentage of white and black tiles is large (e.g., $\rho_B = 34\,\%$, $\rho_w = 66\,\% \rightarrow \rho_N^* \approx 0.52$), while in a relatively difficult task, the difference is small (e.g., $\rho_B = 48\,\%$, $\rho_w = 52\,\% \rightarrow \rho_N^* \approx 0.92$).

At the end of a successful run, all robots will have the same opinion corresponding to the most frequent color.

4.2 Simulation environment

The simulations were executed in discrete time steps (ticks)—with 10 ticks per second using the ARGoS robot swarm simulator [18] (version 3.0.0-beta48) and the ARGoS-Epuck [9] plugin with $N = 20$ e-puck robots on a computer cluster. For each experimental run, two nodes were used on the cluster with 16 cores each. Each core has a clock rate of 2.0 GHz and 1 GB of RAM. ARGoS was executed using $N = 20$ threads, and, for the blockchain approach, one *geth* process (an interface for running a full Ethereum node) was started for each robot using a single thread. The simulated robots were programmed to use the IPC socket of *geth*.

Robots can only communicate with each other if there is no other robot hindering the communication and if the robots' distance to each other is smaller than $d_n = 50$ cm. We used this design constraint in order to mimic the capabilities of IR/Bluetooth/RF transmission, so that we can easily test the algorithm using real robots in future research.

4.3 Classical approach

In the classical approach of Valentini et al. [25], the behavior of each robot is determined by a probabilistic finite state machine (PFSM) with Exploration states E_i and Dissemination states D_i ($i \in$ {black, white}, i indicating the current opinion of the robot). In the beginning of the experiment, all robots are in one of the Exploration states.

There are three low-level control routines: (i) the random walk routine, (ii) the obstacle avoidance routine, and (iii) the quality estimation routine. Their execution is dictated by the state of the robot (E_i or D_i).

When in the E_i state, a robot senses the features of the environment; the robot remains in the E_i state for a time determined by drawing a sample from an exponential distribution with mean $\sigma = 100$ ticks when the robot enters the state. In this state, the robot executes the low-level routines (i), (ii), and (iii). Each robot has a current opinion $i \in$ {black, white} about what it believes to be the most frequent tile color. In the quality estimation routine the robot senses the color of the surface once per tick via its ground sensors. It updates its current quality estimate $\hat{\rho}_i$ by calculating the ratio between the number of ticks when it sensed the color of its current opinion and the total number of ticks in the current exploration state. At the end of each exploration state, the robot switches to the dissemination state D_i that matches its opinion i.

When in the D_i state, a robot only executes the low-level routines (i) and (ii)[3]. Additionally, it disseminates its opinion. At the end of the state, it applies a decision-making strategy to decide whether or not to change its opinion. The decision-making strategies considered in [25] are: (i) $DMVD_N$ (voter model): adopt the opinion of a random neighbor; (ii) $DMMD_N$ (majority voting): adopt the opinion of the majority of the neighbors (including the robot's own opinion); (iii) DC_N (direct comparison): adopt the opinion of a random neighbor only if the robot's current quality estimate $\hat{\rho}_i$ is lower than the neighbor's quality estimate.

The used strategy also determines the duration of the state: using the $DMVD_N$ or $DMMD_N$ strategy, the robot selects the duration of its D_i state based on its current quality estimate $\hat{\rho}_i$ by drawing a sample from an exponential distribution with mean $\hat{\rho}_i g$ (the parameter g is a design parameter, which in our experiments was set to $g = 100$ ticks). The duration of the D_i state using the DC_N strategy is independent of the current quality estimate; it is chosen by drawing a sample from an exponential distribution with mean $\rho_{ref} g$.

Only in the last 30 ticks of the state, the robot receives opinions of other robots. It uses the last two received opinions to decide whether or not to change its opinion i by using one of the strategies. Finally, the robot switches to the E_i state.

4.4 Blockchain approach

We designed the blockchain approach[3] to be as similar as possible to the classical approach. The behavior of the robots is determined by a PFSM with the same low-level routines—while respecting particularities of blockchain technology and adding safety measurements for identifying Byzantine robots. Each robot keeps a separate copy of the blockchain and acts as a node and miner in the blockchain network.

The blockchain approach is driven by a blockchain-based smart contract—programming code that is executed and verified via blockchain technology by every node of the blockchain network. The blockchain serves as a medium to share knowledge, record votes, and apply decision-making strategies.

The smart contract provides three functions: `registerRobot`, `applyStrategy`, and `vote`, to initiate their execution, the robots create signed transactions and send them to their peers via the blockchain protocol. The functions do not immediately return a value since the transactions first have to be mined and included into a block. Therefore, the robots listen to *events*, which are created as soon as a transaction is mined.

The experiments are conducted using a private Ethereum network (in contrast to Ethereum's main network). For this purpose, a custom genesis block is used, which allocates 100 ether[4] to each robot; this is enough to ensure that there are no limitations on the number of transactions a robot can send during an experimental run. The mining difficulty is set to a fixed value by modifying Ethereum's source code. This ensures that the mining difficulty is suited for the (simulated) robots' limited computational power and that the experiments are not influenced by Ethereum's automatic adaptation of the mining difficulty.

At the beginning of each experimental run, a *geth* process is started for each robot. Additionally, we introduce an auxiliary *geth* node to which each robot is connected during the initialization phase. The auxiliary node publishes the smart contract and starts to mine in order to include the contract in the blockchain and to obtain its address. Each robot sends a transaction to the `registerRobot` function of the smart contract. This tells the blockchain the robot's public key. The auxiliary node stops its mining process after the sent transactions have been mined. The robots listen to the events created by the `registerRobot` function, which include the robots' initial opinions as well as the block numbers and corresponding block hashes, where the opinions are saved. As soon as all the steps above are successful, the robots disconnect from the auxiliary node and the experimental run is started.

There is no difference between the exploration states of the classical and the blockchain approaches. However, the dissemination states are different due to particularities of the blockchain approach: in the last 30 ticks of the D_i state, a robot connects to the Ethereum processes of its physical neighbors. In addition, in the last 30 ticks of the D_i state, robots are also mining (i.e., they try to find a solution to the Proof-of-Work-based mining puzzle; we used the default Ethereum PoW puzzle). This ensures that the transactions of `applyStrategy` and `vote` are included into the blockchain and distributed in the network. The blockchain consists of different forks (versions) during the experiment since information is often not spread out in the entire network but only in local robot clusters and robots will mine on different versions of the blockchain. Due to the movements of the robots, the network structure of the robots changes over time. Whenever two or more robots are connected to each other, the Ethereum protocol compares the different blockchains and uses the longest chain as the truth. Additionally, transactions that are not yet included in a block are shared. Since the robots have an equal hash rate (computational power), the longest chain will be the one to which most robots contributed.

During the dissemination phase, the robots send transactions to the function `vote`. As in the classical approach, we implemented three decision-making strategies: $DMVD_{bc}$, $DMMD_{bc}$, and DC_{bc}. The amount of votes a robot sends during the dissemination phase depends on the used decision-making/dissemination strategy: when using $DMVD_{bc}$ or $DMMD_{bc}$, the robots create a voting transaction every five ticks of the dissemination state. Therefore, the longer a robot's dissemination time, the more votes it creates. If robots use the DC_{bc} strategy, they create only one voting transaction each time they enter the dissemination state. The voting transaction includes a robot's opinion (and when using the DC_{bc} strategy, also their quality estimate $\hat{\rho}_i$), the number of the stable block[5] their opinion is based upon, and the corresponding block hash (that the robots received when they listened to the events created by `registerRobot`/`applyStrategy`).

The robots interact with `applyStrategy` to obtain their new opinions at the end of the dissemination state. For this purpose, the robots send a transaction with their current opinion as an argument to the function. The `applyStrategy` function then first chooses two pseudo-random opinions (using the block number and public key of the robot as random seed) from the stable block. Choosing the opinions from the stable block increases the probability that the blockchain operates on information on which consensus has been reached and, consequently, reduces the probability that the information was taken from a fork that will be discarded in the future. Then, `applyStrategy` applies the decision-making strategy ($DMVD_{bc}$, $DMMD_{bc}$, or DC_{bc}, depending on the experiment), the logic of the decision-making strategies is implemented in the same way as in the classical approach. The `applyStrategy` function stores the chosen opinion and block number of the stable block together with the public key of the robot in the blockchain.

The robots wait until their `applyStrategy` transaction is mined by a robot of the network. During the waiting time, they connect and disconnect from the Ethereum processes of other robots, continue to mine, perform the random walk while avoiding other robots, but neither disseminate their opinions nor explore the environment. As soon as the transaction is mined and the *event* with the new opinion i is created, they switch to the corresponding E_i state.

The smart contract uses three safety criteria to decide if votes are to be excluded from the blockchain; if at least one of these criteria is met, the voting transaction is ignored. Using the safety criteria, it can, for example, be decided whether the robots have different blockchain versions or whether they were separated from the rest of the swarm for too long: *(i) Outdated opinion*: the opinion of the robot is based on an outdated block number; this is the case if the robot did not send a transaction to `applyStrategy` in the last 25 blocks; *(ii) Contingent exhausted*: after each application of `applyStrategy`, the smart contract assigns a voting contingent to the robot (50 votes when using the $DMMD_{bc}$ or $DMVD_{bc}$ strategy; one vote when using the DC_{bc}); the contingent gets renewed every time the robot sends a transaction to the `applyStrategy` function. This safety criterion prevents "vote spamming"; *(iii) Different blockchain versions*: the robots have different blockchain versions, which is found out by comparing the hash value of the specified blocks. This safety criterion ensures that the opinion of robots is based on information on which consensus has been reached.

4.5 Byzantine robots

We model a Byzantine robot as follows: it always votes for the minority color (black in our experiments) and it keeps a quality estimate of $\hat{\rho}_i = 1.0$, independent of its actual sensor readings. Apart from that, it acts in the same way as a non-Byzantine robot.

To account for Byzantine robots via the smart contract, a robot's public key is added to a blacklist on the blockchain if none of the safety criteria applies and it sends a vote for the color that does not match its stored opinion on the blockchain. In other words, the smart contract detects the inconsistency when a robot votes for a color other than the one that was agreed upon during the consensus process. From this moment on, the votes of this robot are ignored and not added to the list of votes in the blockchain for the remainder of the experimental run. When a robot is identified as a 'malicious' robot, it is with 100 % certainty (i.e., there are no false positives).

4.6 Metrics

To measure the performance of the classical and blockchain approaches we use the following metrics:

Exit probability (E_N): this is "the probability to make the best decision, computed as the proportion of runs that converge to consensus on opinion a." [23]. In our experiments: $a = w$ ('white').
Consensus time of correct outcomes ($T_N^{correct}$): this is the number of seconds needed until all non-Byzantine robots in the swarm have opinion 'white.' The metric is computed over all experimental runs that converged to 'white'; runs that converged to 'black' are not considered.

Experiments

5 EXPERIMENTS

This section describes two experiments in which we compare the classical and the blockchain approaches using the swarm robotics simulator ARGoS. The experiments for the classical approach are conducted by running the code of Valentini et al. [23].[6]

In each experimental run, 50 % of the robots start with opinion 'white' and the other 50 % with opinion 'black' to have an unbiased initial opinion distribution.

5 EXPERIMENTS

This section describes two experiments in which we compare the classical and the blockchain approaches using the swarm robotics simulator ARGoS. The experiments for the classical approach are conducted by running the code of Valentini et al. [23].[6]

In each experimental run, 50 % of the robots start with opinion 'white' and the other 50 % with opinion 'black' to have an unbiased initial opinion distribution.

5.1 Experiment without Byzantine robots

In the first experiment, we vary the difficulty $\rho_b^* = \rho_b/\rho_w$ of the task and compare the classical and blockchain approaches across the different decision-making strategies. The goal of this experiment is to determine if blockchain-based smart contracts can be used for collective decision-making in robot swarms. In this experiment, there are no Byzantine robots.

In the classical approach (Figure 3, top row), the results obtained with the three decision-making strategies show different patterns for the metrics E_N and T_N^{correct}. The DC_{cl} strategy shows the highest exit probability for all difficulty settings. The DMMD_{cl} strategy has an exit probability below 1.0 even for the easiest setting. It drops to chance level at the highest difficulty setting. The DMVD_{cl} strategy is more stable, but its E_N also decreases at higher difficulty settings. For all difficulty settings, the DC_{cl} strategy has the fastest consensus time, followed by DMVD_{cl} and then DMMD_{cl}.

In the blockchain approach (Figure 3, bottom row), the exit probability (E_N) of the DMMD_{bc} and DMVD_{bc} strategy shows a similar pattern and decreases for higher ρ_b^*. The exit probability of the DC_{bc} strategy is $E_N = 1.0$ for all $\rho_b^* \leq 0.79$ and drops to $E_N \approx 0.8$ at the highest difficulty setting. The consensus time using the DMVD_{bc} and DMMD_{bc} strategy is largely unaffected by the difficulty of the task. In contrast, the consensus time of the DC_{bc} strategy rises with the difficulty.

In general, the exit probabilities (E_N) show similar patterns when using the strategies of the classical approach and when using the counterparts of the blockchain approach. However, the DMVD_{bc} performs worse than the DMVD_{cl} for almost all difficulty levels. The consensus times (T_N^{correct}) of the DMVD_{bc} and DMMD_{bc} strategies are, unlike their counterparts in the classical approach, unaffected by the task difficulty. T_N^{correct} of the DMVD_{bc} has a high variability. T_N^{correct} of both the DC_{bc} and DC_{cl} rises with higher difficulties but the DC_{bc} is overall slower.

Discussion. Our results show that similar results can be achieved using the classical approach and the blockchain approach. The performance of the different strategies implemented in the two approaches is not exactly equal. This is due to the fact that some aspects of the classical approach cannot be implemented using the blockchain approach. For example, in the classical approach robots send their opinion directly to their neighbors; if a neighbor has already received the opinion in a previous timestep from the same robot, it is discarded. In contrast, the blockchain approach creates a number of voting transactions proportional to the dissemination time and all votes are stored in the blockchain. Furthermore, mining and waiting for events in the blockchain approach can create delays which are not present in the classical approach.

5.2 Experiment with Byzantine robots

In the second experiment, we test the robustness of the blockchain approach to the presence of Byzantine robots—robots that always vote for black with a quality estimate of $\hat{\rho}_i = 1.0$ (see Section 4.5). Byzantine robots make the collective decision task more difficult. Since they never change their opinion, consensus on the majority color is no longer achievable when one or more Byzantine robots are part of the swarm. Hence, we stop one experimental run as soon as all non-Byzantine robots of the swarm have the same opinion for the first time (*sub-swarm consensus*). This allows for comparisons between the classical and blockchain approaches. In the experiments, we study how increasing the number of Byzantine robots affects the classical and blockchain approaches. We compare the performance of the two approaches for different values of the number k of Byzantine robots in the swarm. The difficulty of the task is set to $\rho_b^* = 0.52$ (34 % of black tiles).

The results (E_N and T_N^{correct}) show a clear difference in the performance of the classical and blockchain approaches (Figure 4) when using the DMVD or DMMD strategy. While the exit probabilities of the classical approach sharply drop below chance level with even a small number of Byzantine robots, the blockchain approach is more resilient and the exit probabilities remain above chance level for almost all values of k.

In contrast, the performance of the DC_{bc} strategy shows a more similar pattern to the DC_{cl} strategy. This pattern occurs since the robots only change their opinion when their current quality estimate is smaller than the selected opinion in the smart contract. Since Byzantine robots send quality estimate $\hat{\rho} = 1.0$, they can always keep their opinion when using the DC_{bc} strategy and act—from the smart contract's perspective—according to the protocol and therefore are not put on the blacklist. In other words, the modulation technique used by the DMVD_{bc} and DMMD_{bc} strategy (sending an amount of votes proportional to the quality estimate $\hat{\rho}$ instead of directly sending the value $\hat{\rho}$) is robust to the presence of Byzantine robots, while the DC_{bc} strategy is vulnerable because Byzantine robots can circumvent the security measurements.

The consensus times of the classical and blockchain approaches significantly differ from each other; this is partly due to the fact that using the classical approach, E_N is small or zero for several values of k. Therefore, $T_\cdot^{\text{correct}}$ is only based on a few runs or cannot be calculated (remember that T_N^{correct} is based on correct outcomes only). The reliability of the consensus time of the classical approach is hence lower because the sample is much smaller. T_N^{correct} of the DC_{cl} strategy is particularly high (with a high variability) at $k \in \{3, 4, 5\}$; for these values of k, the Byzantine robots manage to influence some—but not all—of the non-Byzantine robots, making it difficult to find consensus on any color. In contrast, the DC_{bc} is fast and T_N^{correct} decreases with the number of Byzantine robots.

The slight increase of E_N for $k = 9$ for both the classical and the blockchain approach is due to the small number of remaining non-Byzantine robots: since out of the $N = 20$ robots, ten (non-Byzantine) robots start with initial opinion 'white', there is only one remaining non-Byzantine robot with initial opinion 'black' that has to change its opinion to 'white' to reach sub-swarm consensus.

Discussion. Even with the relaxed sub-swarm consensus metric, the classical approach breaks down with a small number of Byzantine robots. In contrast, the DMVD_{bc} and DMMD_{bc} decision-making strategies yield high exit probabilities since a robot is added to the blacklist whenever there is a mismatch between the opinion it sends to the smart contract residing on the blockchain and its opinion as written in the blockchain.

Experiments

5.1 Experiment without Byzantine robots

In the first experiment, we vary the difficulty $\rho_b^* = \rho_b/\rho_w$ of the task and compare the classical and blockchain approaches across the different decision-making strategies. The goal of this experiment is to determine if blockchain-based smart contracts can be used for collective decision-making in robot swarms. In this experiment, there are no Byzantine robots.

In the classical approach (Figure 3, top row), the results obtained with the three decision-making strategies show different patterns for the metrics E_N and T_N^{correct}. The DC_{cl} strategy shows the highest exit probability for all difficulty settings. The DMMD_{cl} strategy has an exit probability below 1.0 even for the easiest setting. It drops to chance level at the highest difficulty setting. The DMVD_{cl} strategy is more stable, but its E_N also decreases at higher difficulty settings. For all difficulty settings, the DC_{cl} strategy has the fastest consensus time, followed by DMVD_{cl} and then DMMD_{cl}.

In the blockchain approach (Figure 3, bottom row), the exit probability (E_N) of the DMMD_{bc} and DMVD_{bc} strategy shows a similar pattern and decreases for higher ρ_b^*. The exit probability of the DC_{bc} strategy is $E_N = 1.0$ for all $\rho_b^* \leq 0.79$ and drops to $E_N \approx 0.8$ at the highest difficulty setting. The consensus time using the DMVD_{bc} and DMMD_{bc} strategy is largely unaffected by the difficulty of the task. In contrast, the consensus time of the DC_{bc} strategy rises with the difficulty.

In general, the exit probabilities (E_N) show similar patterns when using the strategies of the classical approach and when using the counterparts of the blockchain approach. However, the DMVD_{bc} performs worse than the DMVD_{cl} for almost all difficulty levels. The consensus times (T_N^{correct}) of the DMVD_{bc} and DMMD_{bc} strategies are, unlike their counterparts in the classical approach, unaffected by the task difficulty. T_N^{correct} of the DMVD_{bc} has a high variability.

T_N^{correct} of both the DC_{bc} and DC_{cl} rises with higher difficulties but the DC_{bc} is overall slower.

5 EXPERIMENTS

This section describes two experiments in which we compare the classical and the blockchain approaches using the swarm robotics simulator ARGoS. The experiments for the classical approach are conducted by running the code of Valentini et al. [23].[6]

In each experimental run, 50 % of the robots start with opinion 'white' and the other 50 % with opinion 'black' to have an unbiased initial opinion distribution.

5.1 Experiment without Byzantine robots

In the first experiment, we vary the difficulty $\rho_b^* = \rho_b/\rho_w$ of the task and compare the classical and blockchain approaches across the different decision-making strategies. The goal of this experiment is to determine if blockchain-based smart contracts can be used for collective decision-making in robot swarms. In this experiment, there are no Byzantine robots.

In the classical approach (Figure 3, top row), the results obtained with the three decision-making strategies show different patterns for the metrics E_N and T_N^{correct}. The DC_{cl} strategy shows the highest exit probability for all difficulty settings. The DMMD_{cl} strategy has an exit probability below 1.0 even for the easiest setting. It drops to chance level at the highest difficulty setting. The DMVD_{cl} strategy is more stable, but its E_N also decreases at higher difficulty settings. For all difficulty settings, the DC_{cl} strategy has the fastest consensus time, followed by DMVD_{cl} and then DMMD_{cl}.

In the blockchain approach (Figure 3, bottom row), the exit probability (E_N) of the DMMD_{bc} and DMVD_{bc} strategy shows a similar pattern and decreases for higher ρ_b^*. The exit probability of the DC_{bc} strategy is $E_N = 1.0$ for all $\rho_b^* \leq 0.79$ and drops to $E_N \approx 0.8$ at the highest difficulty setting. The consensus time using the DMVD_{bc} and DMMD_{bc} strategy is largely unaffected by the difficulty of the task. In contrast, the consensus time of the DC_{bc} strategy rises with the difficulty.

In general, the exit probabilities (E_N) show similar patterns when using the strategies of the classical approach and when using the counterparts of the blockchain approach. However, the DMVD_{bc} performs worse than the DMVD_{cl} for almost all difficulty levels. The consensus times (T_N^{correct}) of the DMVD_{bc} and DMMD_{bc} strategies are, unlike their counterparts in the classical approach, unaffected by the task difficulty. T_N^{correct} of the DMVD_{bc} has a high variability.

T_N^{correct} of both the DC_{bc} and DC_{cl} rises with higher difficulties but the DC_{bc} is overall slower.

Discussion. Our results show that similar results can be achieved using the classical approach and the blockchain approach. The performance of the different strategies implemented in the two approaches is not exactly equal. This is due to the fact that some aspects of the classical approach cannot be implemented using the blockchain approach. For example, in the classical approach robots send their opinion directly to their neighbors; if a neighbor has already received the opinion in a previous timestep from the same robot, it is discarded. In contrast, the blockchain approach creates a number of voting transactions proportional to the dissemination time and all votes are stored in the blockchain. Furthermore, mining and waiting for events in the blockchain approach can create delays which are not present in the classical approach.

5.2 Experiment with Byzantine robots

In the second experiment, we test the robustness of the blockchain approach to the presence of Byzantine robots—robots that always vote for black with a quality estimate of $\hat{\rho}_i = 1.0$ (see Section 4.5). Byzantine robots make the collective decision task more difficult. Since they never change their opinion, consensus on the majority color is no longer achievable when one or more Byzantine robots are part of the swarm. Hence, we stop one experimental run as soon as all non-Byzantine robots of the swarm have the same opinion for the first time (*sub-swarm consensus*). This allows for comparisons between the classical and blockchain approaches. In the experiments, we study how increasing the number of Byzantine robots affects the classical and blockchain approaches. We compare the performance of the two approaches for different values of the number k of Byzantine robots in the swarm. The difficulty of the task is set to $\rho_b^* = 0.52$ (34 % of black tiles).

The results (E_N and T_N^{correct}) show a clear difference in the performance of the classical and blockchain approaches (Figure 4) when using the DMVD or DMMD strategy. While the exit probabilities of the classical approach sharply drop below chance level with even a small number of Byzantine robots, the blockchain approach is more resilient and the exit probabilities remain above chance level for almost all values of k.

In contrast, the performance of the DC_{bc} strategy shows a more similar pattern to the DC_{cl} strategy. This pattern occurs since the robots only change their opinion when their current quality estimate is smaller than the selected opinion in the smart contract. Since Byzantine robots send quality estimate $\hat{\rho} = 1.0$, they can always keep their opinion when using the DC_{bc} strategy and act—from the smart contract's perspective—according to the protocol and therefore are not put on the blacklist. In other words, the modulation technique used by the DMVD_{bc} and DMMD_{bc} strategy (sending an amount of votes proportional to the quality estimate $\hat{\rho}$ instead of directly sending the value $\hat{\rho}$) is robust to the presence of Byzantine robots, while the DC_{bc} strategy is vulnerable because Byzantine robots can circumvent the security measurements.

The consensus times of the classical and blockchain approaches significantly differ from each other; this is partly due to the fact that using the classical approach, E_N is small or zero for several values of k. Therefore, T_N^{correct} is only based on a few runs or cannot be calculated (remember that T_N^{correct} is based on correct outcomes only). The reliability of the consensus time of the classical approach is hence lower because the sample is much smaller. T_N^{correct} of the DC_{cl} strategy is particularly high (with a high variability) at $k \in \{3, 4, 5\}$; for these values of k, the Byzantine robots manage to influence some—but not all—of the non-Byzantine robots, making it difficult to find consensus on any color. In contrast, the DC_{bc} is fast and T_N^{correct} decreases with the number of Byzantine robots.

The slight increase of E_N for $k = 9$ for both the classical and the blockchain approach is due to the small number of remaining non-Byzantine robots: since out of the $N = 20$ robots, ten (non-Byzantine) robots start with initial opinion 'white', there is only one remaining non-Byzantine robot with initial opinion 'black' that has to change its opinion to 'white' to reach sub-swarm consensus.

Discussion. Even with the relaxed sub-swarm consensus metric, the classical approach breaks down with a small number of Byzantine robots. In contrast, the DMVD_{bc} and DMMD_{bc} decision-making strategies yield high exit probabilities since a robot is added to the blacklist whenever there is a mismatch between the opinion it sends to the smart contract residing on the blockchain and its opinion as written in the blockchain.

Experiments

Discussion. Our results show that similar results can be achieved using the classical approach and the blockchain approach. The performance of the different strategies implemented in the two approaches is not exactly equal. This is due to the fact that some aspects of the classical approach cannot be implemented using the blockchain approach. For example, in the classical approach robots send their opinion directly to their neighbors; if a neighbor has already received the opinion in a previous timestep from the same robot, it is discarded. In contrast, the blockchain approach creates a number of voting transactions proportional to the dissemination time and all votes are stored in the blockchain. Furthermore, mining and waiting for events in the blockchain approach can create delays which are not present in the classical approach.

5 EXPERIMENTS

This section describes two experiments in which we compare the classical and the blockchain approaches using the swarm robotics simulator ARGoS. The experiments for the classical approach are conducted by running the code of Valentini et al. [23].[6]

In each experimental run, 50 % of the robots start with opinion 'white' and the other 50 % with opinion 'black' to have an unbiased initial opinion distribution.

5.1 Experiment without Byzantine robots

In the first experiment, we vary the difficulty $\rho_b^* = \rho_b/\rho_w$ of the task and compare the classical and blockchain approaches across the different decision-making strategies. The goal of this experiment is to determine if blockchain-based smart contracts can be used for collective decision-making in robot swarms. In this experiment, there are no Byzantine robots.

In the classical approach (Figure 3, top row), the results obtained with the three decision-making strategies show different patterns for the metrics E_N and T_N^{correct}. The DC_{cl} strategy shows the highest exit probability for all difficulty settings. The DMMD_{cl} strategy has an exit probability below 1.0 even for the easiest setting. It drops to chance level at the highest difficulty setting. The DMVD_{cl} strategy is more stable, but its E_N also decreases at higher difficulty settings. For all difficulty settings, the DC_{cl} strategy has the fastest consensus time, followed by DMVD_{cl} and then DMMD_{cl}.

In the blockchain approach (Figure 3, bottom row), the exit probability (E_N) of the DMMD_{bc} and DMVD_{bc} strategy shows a similar pattern and decreases for higher ρ_b^*. The exit probability of the DC_{bc} strategy is $E_N = 1.0$ for all $\rho_b^* \leq 0.79$ and drops to $E_N \approx 0.8$ at the highest difficulty setting. The consensus time using the DMVD_{bc} and DMMD_{bc} strategy is largely unaffected by the difficulty of the task. In contrast, the consensus time of the DC_{bc} strategy rises with the difficulty.

In general, the exit probabilities (E_N) show similar patterns when using the strategies of the classical approach and when using the counterparts of the blockchain approach. However, the DMVD_{bc} performs worse than the DMVD_{cl} for almost all difficulty levels. The consensus times (T_N^{correct}) of the DMVD_{bc} and DMMD_{bc} strategies are, unlike their counterparts in the classical approach, unaffected by the task difficulty. T_N^{correct} of the DMVD_{bc} has a high variability.

T_N^{correct} of both the DC_{bc} and DC_{cl} rises with higher difficulties but the DC_{bc} is overall slower.

Discussion. Our results show that similar results can be achieved using the classical approach and the blockchain approach. The performance of the different strategies implemented in the two approaches is not exactly equal. This is due to the fact that some aspects of the classical approach cannot be implemented using the blockchain approach. For example, in the classical approach robots send their opinion directly to their neighbors; if a neighbor has already received the opinion in a previous timestep from the same robot, it is discarded. In contrast, the blockchain approach creates a number of voting transactions proportional to the dissemination time and all votes are stored in the blockchain. Furthermore, mining and waiting for events in the blockchain approach can create delays which are not present in the classical approach.

5.2 Experiment with Byzantine robots

In the second experiment, we test the robustness of the blockchain approach to the presence of Byzantine robots—robots that always vote for black with a quality estimate of $\hat{\rho}_i = 1.0$ (see Section 4.5). Byzantine robots make the collective decision task more difficult. Since they never change their opinion, consensus on the majority color is no longer achievable when one or more Byzantine robots are part of the swarm. Hence, we stop one experimental run as soon as all non-Byzantine robots of the swarm have the same opinion for the first time (*sub-swarm consensus*). This allows for comparisons between the classical and blockchain approaches. In the experiments, we study how increasing the number of Byzantine robots affects the classical and blockchain approaches. We compare the performance of the two approaches for different values of the number k of Byzantine robots in the swarm. The difficulty of the task is set to $\rho_b^* = 0.52$ (34 % of black tiles).

The results (E_N and T_N^{correct}) show a clear difference in the performance of the classical and blockchain approaches (Figure 4) when using the DMVD or DMMD strategy. While the exit probabilities of the classical approach sharply drop below chance level with even a small number of Byzantine robots, the blockchain approach is more resilient and the exit probabilities remain above chance level for almost all values of k.

In contrast, the performance of the DC_{bc} strategy shows a more similar pattern to the DC_{cl} strategy. This pattern occurs since the robots only change their opinion when their current quality estimate is smaller than the selected opinion in the smart contract. Since Byzantine robots send quality estimate $\hat{\rho} = 1.0$, they can always keep their opinion when using the DC_{bc} strategy and act—from the smart contract's perspective—according to the protocol and therefore are not put on the blacklist. In other words, the modulation technique used by the DMVD_{bc} and DMMD_{bc} strategy (sending an amount of votes proportional to the quality estimate $\hat{\rho}$ instead of directly sending the value $\hat{\rho}$) is robust to the presence of Byzantine robots, while the DC_{bc} strategy is vulnerable because Byzantine robots can circumvent the security measurements.

The consensus times of the classical and blockchain approaches significantly differ from each other; this is partly due to the fact that using the classical approach, E_N is small or zero for several values of k. Therefore, T_N^{correct} is only based on a few runs or cannot be calculated (remember that T_N^{correct} is based on correct outcomes only). The reliability of the consensus time of the classical approach is hence lower because the sample is much smaller. T_N^{correct} of the DC_{cl} strategy is particularly high (with a high variability) at $k \in \{3, 4, 5\}$; for these values of k, the Byzantine robots manage to influence some—but not all—of the non-Byzantine robots, making it difficult to find consensus on any color. In contrast, the DC_{bc} is fast and T_N^{correct} decreases with the number of Byzantine robots.

The slight increase of E_N for $k = 9$ for both the classical and the blockchain approach is due to the small number of remaining non-Byzantine robots: since out of the $N = 20$ robots, ten (non-Byzantine) robots start with initial opinion 'white', there is only one remaining non-Byzantine robot with initial opinion 'black' that has to change its opinion to 'white' to reach sub-swarm consensus.

Discussion. Even with the relaxed sub-swarm consensus metric, the classical approach breaks down with a small number of Byzantine robots. In contrast, the DMVD_{bc} and DMMD_{bc} decision-making strategies yield high exit probabilities since a robot is added to the blacklist whenever there is a mismatch between the opinion it sends to the smart contract residing on the blockchain and its opinion as written in the blockchain.

Experiments

5.2 Experiment with Byzantine robots

In the second experiment, we test the robustness of the blockchain approach to the presence of Byzantine robots—robots that always vote for black with a quality estimate of $\hat{\rho}_i = 1.0$ (see Section 4.5). Byzantine robots make the collective decision task more difficult. Since they never change their opinion, consensus on the majority color is no longer achievable when one or more Byzantine robots are part of the swarm. Hence, we stop one experimental run as soon as all non-Byzantine robots of the swarm have the same opinion for the first time (*sub-swarm consensus*). This allows for comparisons between the classical and blockchain approaches. In the experiments, we study how increasing the number of Byzantine robots affects the classical and blockchain approaches. We compare the performance of the two approaches for different values of the number k of Byzantine robots in the swarm. The difficulty of the task is set to $\rho_b^* = 0.52$ (34 % of black tiles).

The results (E_N and T_N^{correct}) show a clear difference in the performance of the classical and blockchain approaches (Figure 4) when using the DMVD or DMMD strategy. While the exit probabilities of the classical approach sharply drop below chance level with even a small number of Byzantine robots, the blockchain approach is more resilient and the exit probabilities remain above chance level for almost all values of k.

In contrast, the performance of the DC_{bc} strategy shows a more similar pattern to the DC_{cl} strategy. This pattern occurs since the robots only change their opinion when their current quality estimate is smaller than the selected opinion in the smart contract. Since Byzantine robots send quality estimate $\hat{\rho} = 1.0$, they can always keep their opinion when using the DC_{bc} strategy and act—from the smart contract's perspective—according to the protocol and therefore are not put on the blacklist. In other words, the modulation technique used by the DMVD_{bc} and DMMD_{bc} strategy (sending an amount of votes proportional to the quality estimate $\hat{\rho}$ instead of directly sending the value $\hat{\rho}$) is robust to the presence of Byzantine robots, while the DC_{bc} strategy is vulnerable because Byzantine robots can circumvent the security measurements.

The consensus times of the classical and blockchain approaches significantly differ from each other; this is partly due to the fact that using the classical approach, E_N is small or zero for several values of k. Therefore, T_N^{correct} is only based on a few runs or cannot be calculated (remember that T_N^{correct} is based on correct outcomes only). The reliability of the consensus time of the classical approach is hence lower because the sample is much smaller. T_N^{correct} of the DC_{cl} strategy is particularly high (with a high variability) at $k \in \{3, 4, 5\}$; for these values of k, the Byzantine robots manage to influence some—but not all—of the non-Byzantine robots, making it difficult to find consensus on any color. In contrast, the DC_{bc} is fast and T_N^{correct} decreases with the number of Byzantine robots.

The slight increase of E_N for $k = 9$ for both the classical and the blockchain approach is due to the small number of remaining non-Byzantine robots: since out of the $N = 20$ robots, ten (non-Byzantine) robots start with initial opinion 'white', there is only one remaining non-Byzantine robot with initial opinion 'black' that has to change its opinion to 'white' to reach sub-swarm consensus.

5 EXPERIMENTS

This section describes two experiments in which we compare the classical and the blockchain approaches using the swarm robotics simulator ARGoS. The experiments for the classical approach are conducted by running the code of Valentini et al. [23].[6]

In each experimental run, 50 % of the robots start with opinion 'white' and the other 50 % with opinion 'black' to have an unbiased initial opinion distribution.

5.1 Experiment without Byzantine robots

In the first experiment, we vary the difficulty $\rho_b^* = \rho_b/\rho_w$ of the task and compare the classical and blockchain approaches across the different decision-making strategies. The goal of this experiment is to determine if blockchain-based smart contracts can be used for collective decision-making in robot swarms. In this experiment, there are no Byzantine robots.

In the classical approach (Figure 3, top row), the results obtained with the three decision-making strategies show different patterns for the metrics E_N and T_N^{correct}. The DC_{cl} strategy shows the highest exit probability for all difficulty settings. The DMMD_{cl} strategy has an exit probability below 1.0 even for the easiest setting. It drops to chance level at the highest difficulty setting. The DMVD_{cl} strategy is more stable, but its E_N also decreases at higher difficulty settings. For all difficulty settings, the DC_{cl} strategy has the fastest consensus time, followed by DMVD_{cl} and then DMMD_{cl}.

In the blockchain approach (Figure 3, bottom row), the exit probability (E_N) of the DMMD_{bc} and DMVD_{bc} strategy shows a similar pattern and decreases for higher ρ_b^*. The exit probability of the DC_{bc} strategy is $E_N = 1.0$ for all $\rho_b^* \leq 0.79$ and drops to $E_N \approx 0.8$ at the highest difficulty setting. The consensus time using the DMVD_{bc} and DMMD_{bc} strategy is largely unaffected by the difficulty of the task. In contrast, the consensus time of the DC_{bc} strategy rises with the difficulty.

In general, the exit probabilities (E_N) show similar patterns when using the strategies of the classical approach and when using the counterparts of the blockchain approach. However, the DMVD_{bc} performs worse than the DMVD_{cl} for almost all difficulty levels. The consensus times (T_N^{correct}) of the DMVD_{bc} and DMMD_{bc} strategies are, unlike their counterparts in the classical approach, unaffected by the task difficulty. T_N^{correct} of the DMVD_{bc} has a high variability. T_N^{correct} of both the DC_{bc} and DC_{cl} rises with higher difficulties but the DC_{bc} is overall slower.

Discussion. Our results show that similar results can be achieved using the classical approach and the blockchain approach. The performance of the different strategies implemented in the two approaches is not exactly equal. This is due to the fact that some aspects of the classical approach cannot be implemented using the blockchain approach. For example, in the classical approach robots send their opinion directly to their neighbors; if a neighbor has already received the opinion in a previous timestep from the same robot, it is discarded. In contrast, the blockchain approach creates a number of voting transactions proportional to the dissemination time and all votes are stored in the blockchain. Furthermore, mining and waiting for events in the blockchain approach can create delays which are not present in the classical approach.

5.2 Experiment with Byzantine robots

In the second experiment, we test the robustness of the blockchain approach to the presence of Byzantine robots—robots that always vote for black with a quality estimate of $\hat{\rho}_i = 1.0$ (see Section 4.5). Byzantine robots make the collective decision task more difficult. Since they never change their opinion, consensus on the majority color is no longer achievable when one or more Byzantine robots are part of the swarm. Hence, we stop one experimental run as soon as all non-Byzantine robots of the swarm have the same opinion for the first time (*sub-swarm consensus*). This allows for comparisons between the classical and blockchain approaches. In the experiments, we study how increasing the number of Byzantine robots affects the classical and blockchain approaches. We compare the performance of the two approaches for different values of the number k of Byzantine robots in the swarm. The difficulty of the task is set to $\rho_b^* = 0.52$ (34 % of black tiles).

The results (E_N and T_N^{correct}) show a clear difference in the performance of the classical and blockchain approaches (Figure 4) when using the DMVD or DMMD strategy. While the exit probabilities of the classical approach sharply drop below chance level with even a small number of Byzantine robots, the blockchain approach is more resilient and the exit probabilities remain above chance level for almost all values of k.

In contrast, the performance of the DC_{bc} strategy shows a more similar pattern to the DC_{cl} strategy. This pattern occurs since the robots only change their opinion when their current quality estimate is smaller than the selected opinion in the smart contract. Since Byzantine robots send quality estimate $\hat{\rho} = 1.0$, they can always keep their opinion when using the DC_{bc} strategy and act—from the smart contract's perspective—according to the protocol and therefore are not put on the blacklist. In other words, the modulation technique used by the DMVD_{bc} and DMMD_{bc} strategy (sending an amount of votes proportional to the quality estimate $\hat{\rho}$ instead of directly sending the value $\hat{\rho}$) is robust to the presence of Byzantine robots, while the DC_{bc} strategy is vulnerable because Byzantine robots can circumvent the security measurements.

The consensus times of the classical and blockchain approaches significantly differ from each other; this is partly due to the fact that using the classical approach, E_N is small or zero for several values of k. Therefore, T_N^{correct} is only based on a few runs or cannot be calculated (remember that T_N^{correct} is based on correct outcomes only). The reliability of the consensus time of the classical approach is hence lower because the sample is much smaller. T_N^{correct} of the DC_{cl} strategy is particularly high (with a high variability) at $k \in \{3, 4, 5\}$; for these values of k, the Byzantine robots manage to influence some—but not all—of the non-Byzantine robots, making it difficult to find consensus on any color. In contrast, the DC_{bc} is fast and T_N^{correct} decreases with the number of Byzantine robots.

The slight increase of E_N for $k = 9$ for both the classical and the blockchain approach is due to the small number of remaining non-Byzantine robots: since out of the $N = 20$ robots, ten (non-Byzantine) robots start with initial opinion 'white', there is only one remaining non-Byzantine robot with initial opinion 'black' that has to change its opinion to 'white' to reach sub-swarm consensus.

Discussion. Even with the relaxed sub-swarm consensus metric, the classical approach breaks down with a small number of Byzantine robots. In contrast, the DMVD_{bc} and DMMD_{bc} decision-making strategies yield high exit probabilities since a robot is added to the blacklist whenever there is a mismatch between the opinion it sends to the smart contract residing on the blockchain and its opinion as written in the blockchain.

Experiments

Discussion. Even with the relaxed sub-swarm consensus metric, the classical approach breaks down with a small number of Byzantine robots. In contrast, the DMVD_{bc} and DMMD_{bc} decision-making strategies yield high exit probabilities since a robot is added to the blacklist whenever there is a mismatch between the opinion it sends to the smart contract residing on the blockchain and its opinion as written in the blockchain.

5 EXPERIMENTS

This section describes two experiments in which we compare the classical and the blockchain approaches using the swarm robotics simulator ARGoS. The experiments for the classical approach are conducted by running the code of Valentini et al. [23].[6]

In each experimental run, 50 % of the robots start with opinion 'white' and the other 50 % with opinion 'black' to have an unbiased initial opinion distribution.

5.1 Experiment without Byzantine robots

In the first experiment, we vary the difficulty $\rho_b^* = \rho_b/\rho_w$ of the task and compare the classical and blockchain approaches across the different decision-making strategies. The goal of this experiment is to determine if blockchain-based smart contracts can be used for collective decision-making in robot swarms. In this experiment, there are no Byzantine robots.

In the classical approach (Figure 3, top row), the results obtained with the three decision-making strategies show different patterns for the metrics E_N and T_N^{correct}. The DC_{cl} strategy shows the highest exit probability for all difficulty settings. The DMMD_{cl} strategy has an exit probability below 1.0 even for the easiest setting. It drops to chance level at the highest difficulty setting. The DMVD_{cl} strategy is more stable, but its E_N also decreases at higher difficulty settings. For all difficulty settings, the DC_{cl} strategy has the fastest consensus time, followed by DMVD_{cl} and then DMMD_{cl}.

In the blockchain approach (Figure 3, bottom row), the exit probability (E_N) of the DMMD_{bc} and DMVD_{bc} strategy shows a similar pattern and decreases for higher ρ_b^*. The exit probability of the DC_{bc} strategy is $E_N = 1.0$ for all $\rho_b^* \leq 0.79$ and drops to $E_N \approx 0.8$ at the highest difficulty setting. The consensus time using the DMVD_{bc} and DMMD_{bc} strategy is largely unaffected by the difficulty of the task. In contrast, the consensus time of the DC_{bc} strategy rises with the difficulty.

In general, the exit probabilities (E_N) show similar patterns when using the strategies of the classical approach and when using the counterparts of the blockchain approach. However, the DMVD_{bc} performs worse than the DMVD_{cl} for almost all difficulty levels. The consensus times (T_N^{correct}) of the DMVD_{bc} and DMMD_{bc} strategies are, unlike their counterparts in the classical approach, unaffected by the task difficulty. T_N^{correct} of the DMVD_{bc} has a high variability. T_N^{correct} of both the DC_{bc} and DC_{cl} rises with higher difficulties but the DC_{bc} is overall slower.

Discussion. Our results show that similar results can be achieved using the classical approach and the blockchain approach. The performance of the different strategies implemented in the two approaches is not exactly equal. This is due to the fact that some aspects of the classical approach cannot be implemented using the blockchain approach. For example, in the classical approach robots send their opinion directly to their neighbors; if a neighbor has already received the opinion in a previous timestep from the same robot, it is discarded. In contrast, the blockchain approach creates a number of voting transactions proportional to the dissemination time and all votes are stored in the blockchain. Furthermore, mining and waiting for events in the blockchain approach can create delays which are not present in the classical approach.

5.2 Experiment with Byzantine robots

In the second experiment, we test the robustness of the blockchain approach to the presence of Byzantine robots—robots that always vote for black with a quality estimate of $\hat{\rho}_i = 1.0$ (see Section 4.5). Byzantine robots make the collective decision task more difficult. Since they never change their opinion, consensus on the majority color is no longer achievable when one or more Byzantine robots are part of the swarm. Hence, we stop one experimental run as soon as all non-Byzantine robots of the swarm have the same opinion for the first time (*sub-swarm consensus*). This allows for comparisons between the classical and blockchain approaches. In the experiments, we study how increasing the number of Byzantine robots affects the classical and blockchain approaches. We compare the performance of the two approaches for different values of the number k of Byzantine robots in the swarm. The difficulty of the task is set to $\rho_b^* = 0.52$ (34 % of black tiles).

The results (E_N and T_N^{correct}) show a clear difference in the performance of the classical and blockchain approaches (Figure 4) when using the DMVD or DMMD strategy. While the exit probabilities of the classical approach sharply drop below chance level with even a small number of Byzantine robots, the blockchain approach is more resilient and the exit probabilities remain above chance level for almost all values of k.

In contrast, the performance of the DC_{bc} strategy shows a more similar pattern to the DC_{cl} strategy. This pattern occurs since the robots only change their opinion when their current quality estimate is smaller than the selected opinion in the smart contract. Since Byzantine robots send quality estimate $\hat{\rho} = 1.0$, they can always keep their opinion when using the DC_{bc} strategy and act—from the smart contract's perspective—according to the protocol and therefore are not put on the blacklist. In other words, the modulation technique used by the DMVD_{bc} and DMMD_{bc} strategy (sending an amount of votes proportional to the quality estimate $\hat{\rho}$ instead of directly sending the value $\hat{\rho}$) is robust to the presence of Byzantine robots, while the DC_{bc} strategy is vulnerable because Byzantine robots can circumvent the security measurements.

The consensus times of the classical and blockchain approaches significantly differ from each other; this is partly due to the fact that using the classical approach, E_N is small or zero for several values of k. Therefore, T_N^{correct} is only based on a few runs or cannot be calculated (remember that T_N^{correct} is based on correct outcomes only). The reliability of the consensus time of the classical approach is hence lower because the sample is much smaller. T_N^{correct} of the DC_{cl} strategy is particularly high (with a high variability) at $k \in \{3, 4, 5\}$; for these values of k, the Byzantine robots manage to influence some—but not all—of the non-Byzantine robots, making it difficult to find consensus on any color. In contrast, the DC_{bc} is fast and T_N^{correct} decreases with the number of Byzantine robots.

The slight increase of E_N for $k = 9$ for both the classical and the blockchain approach is due to the small number of remaining non-Byzantine robots: since out of the $N = 20$ robots, ten (non-Byzantine) robots start with initial opinion 'white', there is only one remaining non-Byzantine robot with initial opinion 'black' that has to change its opinion to 'white' to reach sub-swarm consensus.

Discussion. Even with the relaxed sub-swarm consensus metric, the classical approach breaks down with a small number of Byzantine robots. In contrast, the DMVD_{bc} and DMMD_{bc} decision-making strategies yield high exit probabilities since a robot is added to the blacklist whenever there is a mismatch between the opinion it sends to the smart contract residing on the blockchain and its opinion as written in the blockchain.

Experiments

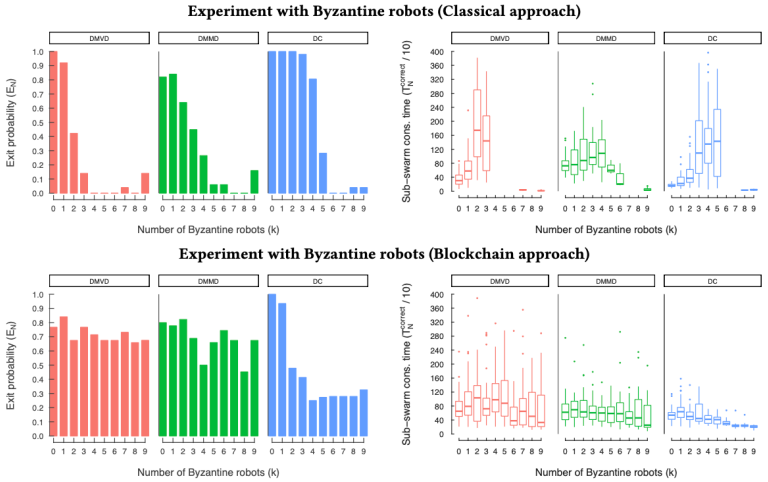


Figure 4: Exit probability (E_N) and consensus time of correct outcomes (T_N^{correct}) as a function of the number of Byzantine robots (top row: classical approach; bottom row: blockchain approach). Since T_N^{correct} is calculated using correct outcomes only, it is only based on a few runs or cannot be calculated when E_N is small or zero. The data is collected by executing 50 repetitions for each combination of number of Byzantine robots and decision-making strategy.

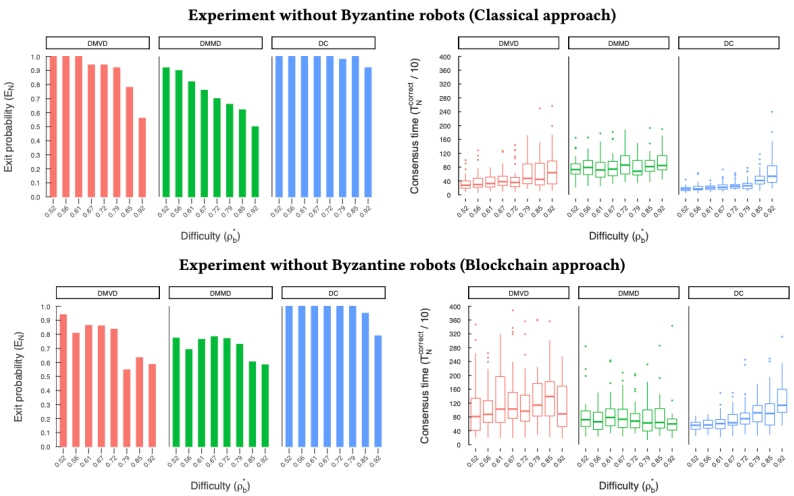


Figure 3: Exit probability (E_N) and consensus time of correct outcomes (T_N^{correct}) as a function of the difficulty of the task $\rho_b^* \in \{0.52, 0.56, 0.61, 0.67, 0.72, 0.79, 0.85, 0.92\}$ (top row: classical approach; bottom row: blockchain approach). The data is collected by executing 40 repetitions for each combination of difficulty level and decision-making strategy.

Discussion

6 DISCUSSION

In this section, we list the advantages and disadvantages of the classical and blockchain approaches and discuss the results of the experiments in more general terms.

In presence of Byzantine robots, the classical approach always converges to the wrong color if the simulation is not stopped when *sub-swarm consensus* is reached. In the blockchain approach, in contrast, consensus could be achieved in a fully decentralized way via the smart contract, without a priori knowledge regarding which robots are Byzantine. However, we considered sub-swarm consensus measured by an external observer. Even though this metric cannot be measured in a real robot deployment (as we would not know which robots are Byzantine), we use it to see whether the "blockchain machinery" causes the convergence of the non-Byzantine robots to be much slower compared to the classical approach.

The Proof-of-Work secures the data of the robot swarm as long as no intruders with significantly higher hash rates get access to the blockchain. However, the work presented in this paper is a proof-of-concept and in future work we will consider other consensus protocols, such as Proof-of-Stake (already implemented in some existing blockchain protocols), Proof-of-Sensing (only robots that can produce a certain sensory output can send/validate transactions), or even Proof-of-*physical*-Work (only robots that can prove that they have performed physical work—such as collecting an item—can send/validate transactions).

Using the classical approach, the message size is 2 Bytes with the $\mathrm{DMVD_{cl}}$/$\mathrm{DMMD_{cl}}$ strategies and 4 Bytes with the $\mathrm{DC_{cl}}$ strategy. The message size in the blockchain approach is significantly larger: approximately 160 Bytes per transaction since it contains also metadata, such as the digital signature and address of the receiving smart contract. For a run of 15 min, the blockchain size is approximately 10 MB for the $\mathrm{DMVD_{bc}}$/$\mathrm{DMMD_{bc}}$ strategies, and approximately 4 MB for the $\mathrm{DC_{bc}}$ strategy. The size increases linearly both with time and number of robots.

We assume that the robots are able to send at least some kB/s. Due to digital signatures in blockchain technology, noisy communication channels will not alter the sent transactions: they will either be received completely or be invalid. Since transactions are distributed in a peer-to-peer manner, unreliable communication channels could also receive past information from different robots after the information has been disseminated in the network.

We have not investigated how sparse connectivity affects the blockchain approach. However, we still expect good performance since all crucial actions of different robot clusters get recorded and a new global view could be obtained by merging the different views of the clusters once reunited.

Conclusions and Future Work

7 CONCLUSIONS AND FUTURE WORK

While swarm robotics systems are often claimed to be highly fault-tolerant, in some cases one or a few malfunctioning robots suffice to make a robot swarm unable to continue operating. To the best of our knowledge, in this paper we have described, implemented, and evaluated the first proof-of-concept of a robot swarm that manages malfunctioning robots via blockchain technology. Using a blockchain-based smart contract, we demonstrated that Byzantine robots can be identified and excluded from the swarm.

Until now, blockchain technology was mainly used on the Internet with communication gaps of a few seconds. Its use in swarm robotics poses several challenges. For example, the communication can be much slower and the information only locally available for longer time periods (i.e., in local robot clusters that are separated from the rest of the swarm). Moreover, the hardware used in swarm robotics is usually much more limited (computational/memory limitations) than the hardware used in desktop computers or computing clusters.

In future work, we will expand the range of possible Byzantine failures and will study how blockchain technology can be used in other swarm robotics tasks. Additionally, we intend to scale down the computation and memory requirements to make the blockchain run on devices with very limited computational capabilities. Finally, we are currently working on transferring the system to a swarm of real e-puck robots.

Acknowledgement

8 ACKNOWLEDGEMENTS

Volker Strobel and Marco Dorigo acknowledge support from the Belgian F.R.S.-FNRS and from the FLAG-ERA project RoboCom++. Eduardo Castelló Ferrer acknowledges support from the Marie Skłodowska-Curie actions (EU project BROS - DLV-751615).

Thanks