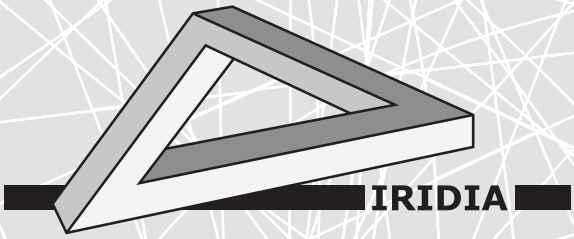

Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

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

V. Strobel, E. Castelló Ferrer, and M. Dorigo

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2017-013

December 2017
Last revision: February 2018

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:

IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2017-013

Revision history:

TR/IRIDIA/2017-013.001 December 2017
TR/IRIDIA/2017-013.002 February 2018

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

Volker Strobel*[1], Eduardo Castelló Ferrer†[2], Marco Dorigo‡[1]

[1]IRIDIA, Université libre de Bruxelles, Brussels, Belgium
[2]MIT Media Lab, Cambridge, Massachusetts, U.S.

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 a few Byzantine robots might 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 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.

Keywords

swarm robotics; blockchain technology; Byzantine robot fault-tolerance

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 [2]. In these environments the robots can usually only communicate in a peer-to-peer manner via noisy and unreliable communication channels and a central source of information could 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) [20].

While swarm robotics systems are often claimed to be highly fault-tolerant, in some cases one or a few *Byzantine* robots—robots that show arbitrarily faulty or malicious behavior—might suffice to let current coordination mechanisms fail [11]. 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 [10]. 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. [10] 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 propagated 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.

*vstrobel@ulb.ac.be
†ecstll@media.mit.edu
‡mdorigo@ulb.ac.be

In this paper, we show that *blockchain technology* provides solutions to the above mentioned 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 [13], recently there have been proposal 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 [4, 21]. 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 3 [14], 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. [19] (*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 to take care of the presence of Byzantine robots. Our blockchain approach also allows to log events in a tamper-proof way: these logs can then be used to analyze, if necessary, 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 do data forensics on decentralized systems such as robot swarms. In the simulations, we 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 [19] 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 fundamental concepts of blockchain technology. Section 4 contrasts the logic of the classical and blockchain approaches. Section 5 evaluates the performance of the approaches by experiments in simulation. Section 6 presents our conclusions and gives directions for future work.

2 Related Work

Many studies (e.g., [3, 15, 17, 12, 9]) have addressed collective decision-making in robot swarms, a key task for the advancement of swarm robotics [2]. Collective decision-making tasks can be divided into two sub-classes: task allocation and consensus achievement [2]. In task allocation, the goal of the swarm is to assign the members to different tasks, so that they maximize the overall swarm performance. In consensus achievement, the goal of the swarm is to agree upon the best among a set of alternatives. The experimental scenario that we use in this paper is based on the work of Valentini et al. [19], who describe and evaluate a collective decision scenario where robots have to determine the most frequent tile color in an environment in which the floor is covered with black and white tiles.

Until now, no protocols for making secure collective decisions in the presence of Byzantine robots have been proposed and security issues in swarm robotics have been largely ignored. Security issues have, to the best of our knowledge, only been studied in [18], where an overview of robot swarms' robustness to different attacker strategies in a cooperative navigation experiment is given; and in [16], where it is described how data privacy can be analyzed in a robot swarm. However, so far, no mechanisms exist to cope with these problems. Zikratov et al. [22] present a reputation management system that assigns a dynamic trust level to robots to identify malicious entities. However, the system allows neither for logging events in a tamper-proof way, nor for solving conflicting situations via a consensus protocol.

The potential use of blockchain technology for managing security issues in robot swarms was first outlined by Ferrer [7]. 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 aims to be the first to provide an actual proof-of-concept of using blockchain technology for the coordination of robot swarms—including a description, the implementation, and evaluation of the approach.

So far, blockchain technology mainly has applications in the financial domain—most notably as a decentralized database for storing transactions of cryptographic tokens (cryptocurrencies). The Internet-of-Things (IoT) domain already uses blockchain technology for managing security and privacy issues. Atzori [1] presents possible control mechanisms, business models, financial systems, micropayments, and the monetization of data as use cases for blockchain technology. Christidis and Devetsikiotis [5] depict how blockchain technology can be used for the automation of workflows with smart contracts and as a marketplace for services between IoT devices. The authors also address drawbacks in blockchain technology, such as the immutability of smart contracts in case of logic/programming errors and the lack of secure communication since the contents of the blockchain can be read by everyone. Dorri et al. [6] describe a lightweight architecture for managing privacy and security challenges of using IoT devices in smart homes.

3 Fundamentals of Blockchain Technology

A blockchain is a distributed database that is replicated among the peers of a network. It is 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 and 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, 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 order, quantity 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 amount of agents.

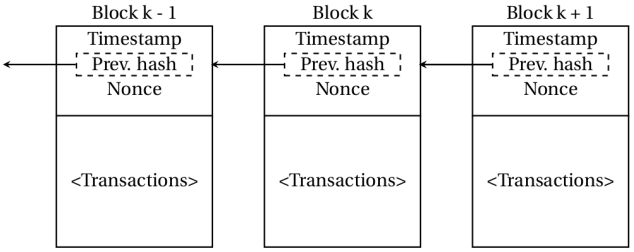

Figure 1: The blockchain—a distributed database—is organized into blocks. Each block consists of two parts, the header and the body. The header contains meta-data for the block—most importantly the hash of the previous block which creates a unique chain of blocks. The body contains the actual data: the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

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 [4]—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 with 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 of the distributed storage, a consensus protocol is used. The consensus protocol serves as a tool for agreeing on the state of the blockchain and for securely extending the information in 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 to add a block to the blockchain. The process of finding such a solution is called *mining*; the nodes that execute that 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 metadata and a changing 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.

The PoW guarantees that changing existing information memorised 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 total network processing power (i.e., 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, the Ethereum protocol [4, 21] introduced blockchain-based smart contracts. Smart contracts are decentralized applications, that can contain deterministic functions and variables that allow for executing arbitrary programming code. For that purpose, the Ethereum protocol uses Solidity[1]: a turing-

[1] `http://solidity.readthedocs.io/en/develop/`, visited on 13 November 2017

complete programming language whose syntax is similar to JavaScript. This language is designed to launch commands on the Ethereum Virtual Machine (EVM). The EVM is the part of the protocol that handles the internal state and the computation components 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."[2]

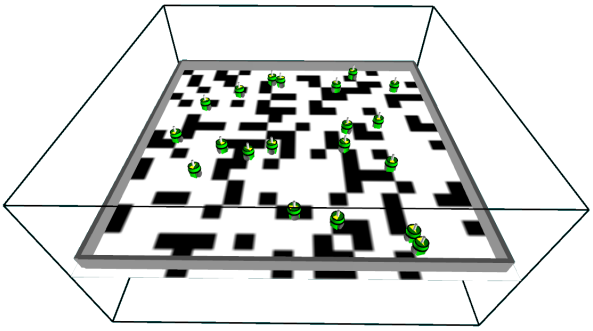

Figure 2: The robots' task is to determine the most frequent color in an environment whose floor is covered with black and white tiles. This collective-decision experiment was conducted using the ARGoS 3 robotics swarm simulator with the plugin for e-puck robots.

4 Methods

In this section, we present Valentini et al.'s work [19] (*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 DC strategy of the blockchain approach).

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 of a black/white grid (Figure 2). Each robot has a current opinion about the correct color, 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: $\rho_b^* = \frac{\rho_b}{\rho_w}$. At the end of a successful run, all robots have the opinion of the majority color (in our experiments it is always the white).

4.2 Simulation environment

The simulations were executed in discrete time steps (ticks)—with 10 ticks per second using the ARGoS 3 robotics swarm simulator [14] and the ARGoS-Epuck [8] 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 random-access memory (RAM). ARGoS 3 was executed using $N = 20$ threads, and, for the blockchain approach, one *geth* process an implementation of the Ethereum protocol, was started for each robot using a single thread. The simulated robots were programmed to use the IPC socket of *geth*.

[2] `https://ethereum.org/`, visited on 22 October 2017

4.3 Classical approach

In the classical approach of Valentini et al. [19], 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 high-level states of the robot (E_i or D_i).

When in the E_i state a robot senses the features of the environment; its duration is determined by drawing a sample from an exponential distribution with mean $\sigma = 100$ ticks when entering the state. In this state, the robot executes the low-level routines (i) - (iii). Each robot has a current opinion $i \in \{\text{black}, \text{white}\}$ about 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 the decision-making strategies to decide whether to change or not its opinion. The decision-making strategies considered in [19] are: (i) $\mathrm{DMVD_{cl}}$ (voter model): adopt the opinion of a random neighbor; (ii) $\mathrm{DMMD_{cl}}$ (majority voting): adopt the opinion of the majority of the neighbors (including the robot's own opinion); (iii) $\mathrm{DC_{cl}}$ (direct comparison): adopt the opinion of a random neighbor only if the robot's current quality estimate $\hat{\rho}_i$ is smaller than the neighbor's quality estimate.

The strategies also determine the duration of the state: using the $\mathrm{DMVD_{cl}}$ or $\mathrm{DMMD_{cl}}$ strategy, the robot uses *direct modulation* and 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 was set to $g = 100$ ticks). The duration of the D_i state using the $\mathrm{DC_{cl}}$ strategy is independent of the current quality estimate (no modulation); 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 and stores them in a list. At deciding whether to change its opinion or not by using one of the strategies, the robot switches to the E_i state.

4.4 Blockchain approach

We designed the blockchain approach 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 full node and miner in the blockchain network.

4.4.1 Smart contract

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 medium for shared knowledge, to record votes, and to 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 have first to be mined and included into a block. Therefore, the robots listen to *events*, which are created as soon as a transaction is mined.

Initialization 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 all robots; this ensures that there are no limitations on the amount of transactions a robot can send during one 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 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.

4.4.2 Control routines

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. The connection is only established if there is no obstacle hindering the communication and if the robots' distance to each other is smaller than $d_n = 50$ cm. We used this design constrain in order to mimic the capabilities of IR/Bluetooth/RF transmission, so we can easily test the algorithm in real robots. In addition, a robot is mining (i.e., it tries to find a solution to the Proof-of-Work-based mining puzzle) in the last 30 ticks of the D_i state. 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. If two or more robots are connected to

[3] This is because when in the dissemination state the goal of the robots is to mix their positions and their opinions.

[4] Ethereum's cryptocurrency (immutable tokens stored on the blockchain)

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, the longest chain will be the one to which most robots contributed.

Interaction 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 $\hat{\rho}_i$) as well as the number of the block, which has exactly $z = 6$ confirmations (*stable block*), 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 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 possibility that the information was just available in a fork. Then, `applyStrategy` applies one of the decision-making strategies; 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 and obstacle avoidance, but neither disseminate their opinions nor explore the environment. As soon as the transaction is mined and the *event* with the new opinion i created, they switch to the corresponding E_i state.

Safety criteria The smart contract uses three safety criteria to decide if votes are to be excluded from the blockchain; if at least one of these cases occur, the voting transaction is ignored. Via the safety criteria, it can, for example, be decided whether 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.
(iii) Different blockchain versions: the robots have different blockchain versions, which is found out by comparing the hash value of the specified blocks.

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, Byzantine robots act in the same way as non-Byzantine robots.

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*). In the experiments, we study how increasing the numbers of Byzantine robots affects the classical and blockchain approaches.

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. 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.

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." [19]. In our experiments: $a = w$ ('white').
Consensus time of correct outcomes ($T_N^{correct}$): this is the number of seconds needed until all robots in the swarm have opinion 'white.' The metric is computed over all ex-

perimental runs that converged to 'white'; runs that converged to 'black' are not considered for this metric.

5 Experiments

This section describes two experiments in which we compare the classical approach with the blockchain approach using the swarm robotics simulator ARGoS 3. The experiments for the classical approach are conducted by running the code of Valentini et al. [19].[5]

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 the 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 of the DMVD_{bc} and DMMD_{bc} strategies are, differently from their counterparts in the classical approach, unaffected by the task difficulty. The T_N^{correct} of the DMVD_{bc} has a high variance. T_N^{correct} of the DC_{bc} and DC_{cl} both rise with higher difficulties but the DC_{bc} is overall slower.

Discussion In this first experiment, our goal was to determine whether blockchain-based smart contracts can be used for collective decision-making in robot swarms. 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 directly implemented using the blockchain approach. For example, in the classical approach robots send their opinion directly to their neighbors; if the 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 hypothesis that our blockchain approach is more fault-tolerant than the classical approach in the presence of Byzantine robots (Section 4.5). 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 approach and the blockchain approach (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 happens because 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 and act—from the smart contract's perspective—according to the protocol and therefore are not put on the blacklist.

The consensus times of the classical and blockchain approaches significantly differ from each other. Using the

[5] We used a different implementation of the DMMD_{cl} strategy, since the original implementation contained a bug that led to the following behavior: if there is a tie in the received opinions (including the robot's own opinion), for example, if robot A has opinion 'black' and receives one vote for 'white' from robot B, robot A always changes to 'white'. Since there were always more white tiles than black tiles in Valentini et al.'s experiments, this resulted in shorter consensus times and higher exit probabilities than what would have been without the bug. Because of this, the results presented in this paper for the DMMD_{cl} strategy are not consistent with those presented in [19].

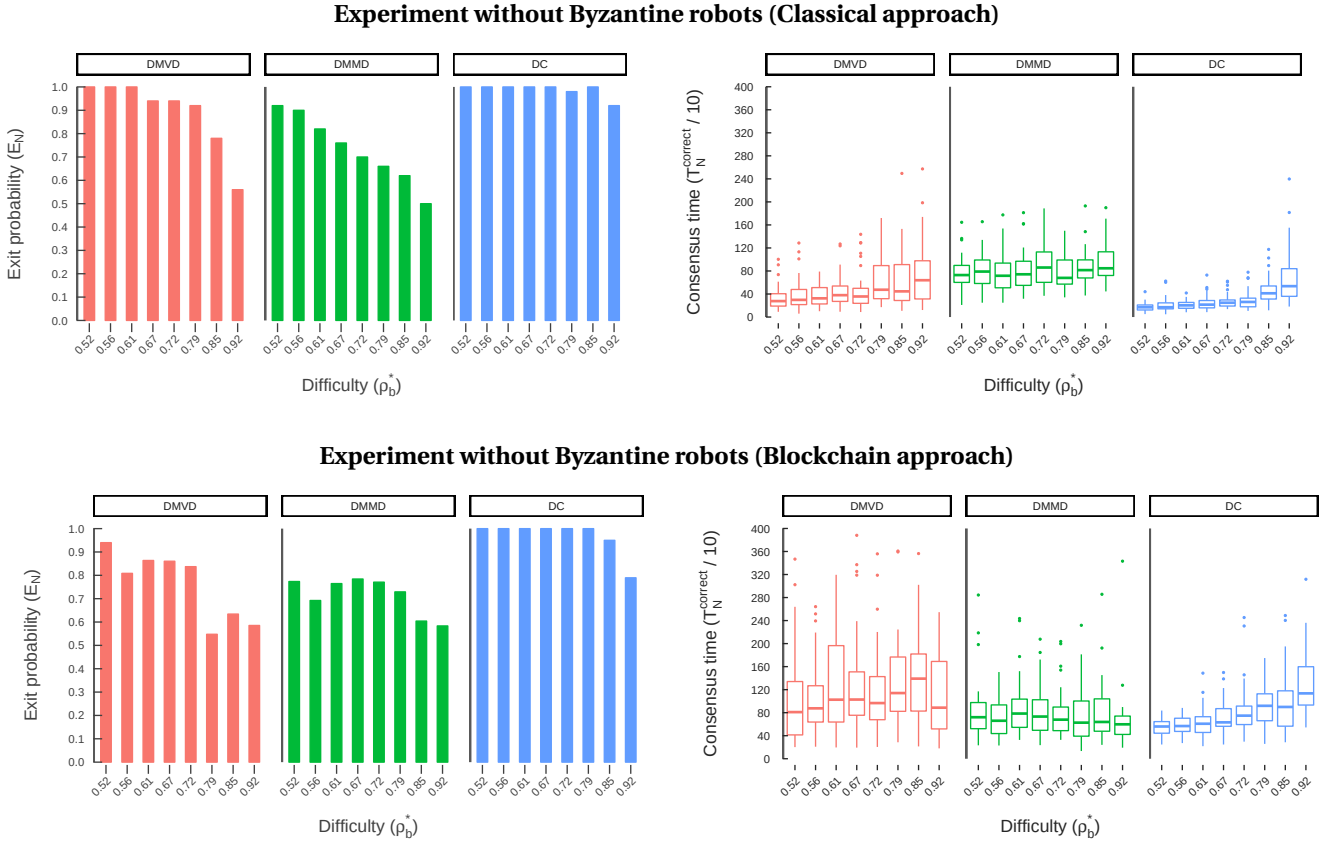

Figure 3: Exit probability (E_N) and consensus time (T_N^{correct}) as a function of the difficulty of the task $\rho_b^* \in \{0.52, 0.56, 0.61, 0.67, 0.72, 0.79, 0.85, 0.92\}$ (top row: classical approach; bottom row: blockchain approach). The data are collected by executing 40 repetitions for each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

classical approach, E_N is small or zero for several values of k , therefore, the T_N^{correct} cannot be calculated or is only based on few runs (T_N^{correct} is based on correct outcomes only). The T_N^{correct} of the DC_{cl} is particularly high (with a high variance) 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 decreases with the number of Byzantine robots.

The slight increase of the exit probability for $k = 9$ for both the classical and the blockchain approach is due to the small number of remaining non-Byzantine robots. If ten 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 reach sub-swarm consensus.

Discussion The classical approach cannot find consensus on the correct outcome if there is at least one Byzantine robot which votes for the minority color (black). In this case, the exit probability would always be zero. We introduced sub-swarm consensus as a new consensus metric, which is achieved as soon as all non-Byzantine robots have the same opinion for the first time. Even with the relaxed sub-swarm consensus metric, the classical approach breaks down even with a small number of Byzantine robots. In contrast, the DMVD_{bc} and DMMD_{bc} decision-making strategies yield high exit probabilities, due to the safety criteria and the blacklist for Byzantine robots.

6 Conclusion 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 system 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 addresses security challenges via blockchain technology.

Using a blockchain-based smart contract, we demonstrated that Byzantine robots can be identified and excluded from the swarm. Moreover, blockchain technology preserves the integrity of transactions while they are prop-

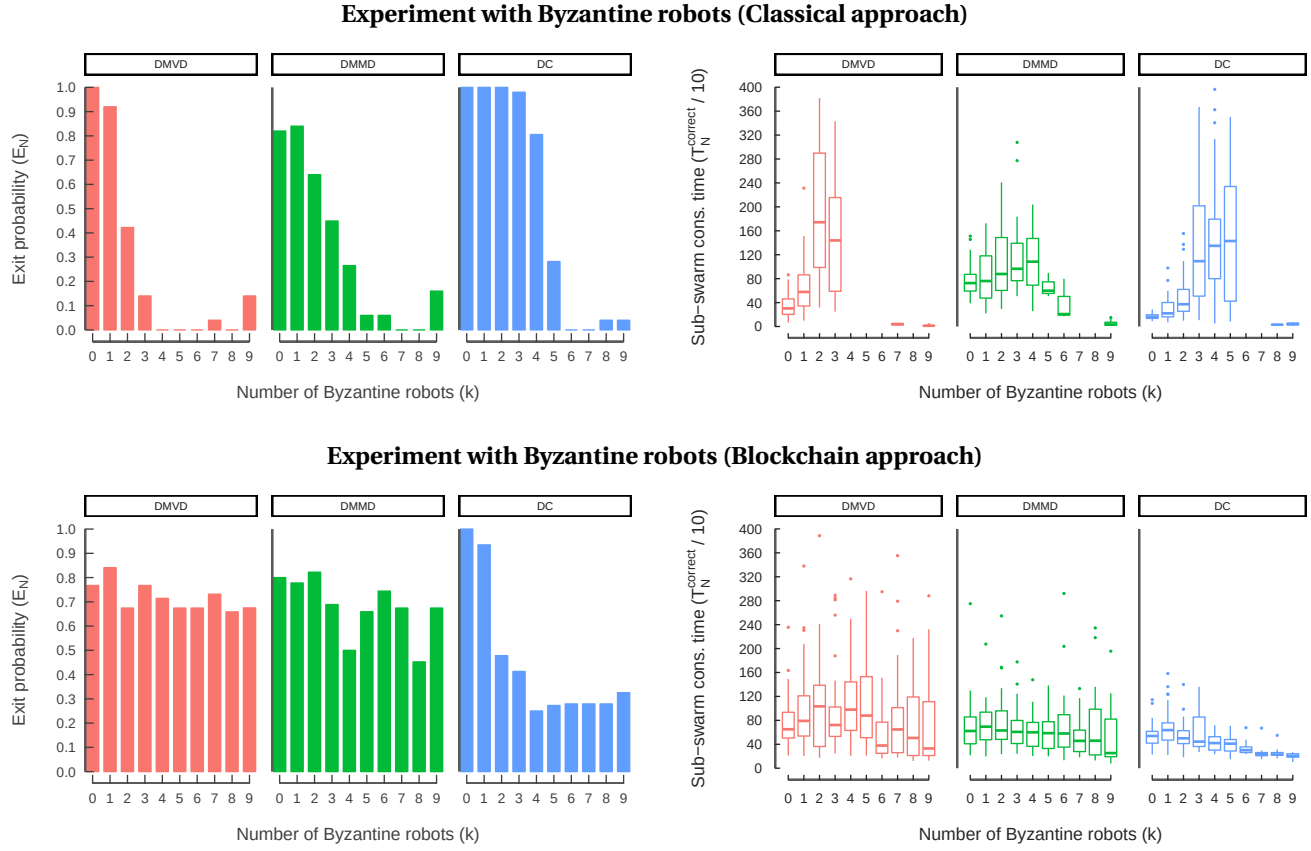

Figure 4: Exit probability (E_N) and consensus time (T_N^{correct}) as a function of the number of Byzantine robots (top row: classical approach; bottom row: blockchain approach). The data are collected by executing 50 repetitions for each combination of number of Byzantine robots and decision-making strategy.

agated through the peer-to-peer network (e.g., a vote for 'black' cannot be changed to a vote for 'white' when being broadcasted through the network) and provides a tool for securely storing critical events in a decentralized log file. Therefore, the transactions sent to the smart contract functions can be analyzed during or after a task, even if several of the swarm members break or get lost.

Until now, blockchain technology was mainly used on the Internet with communication gaps of a few seconds. In swarm robotics, the communication can be much slower, the hardware much more limited (computation/memory limitations), 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). 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. The Proof-of-Work secures the data of the robot swarm as long as no more powerful processors with higher hash rates gets access to the blockchain. We plan to investigate the possibilities of using alternative consensus algorithms, such as *Physical* Proof-of-Work (robots have to execute a task, such as collecting an item, to create new blocks in the blockchain). Additionally, we intend to further scale down the computation and memory requirements to make the blockchain run on devices with very limited computational capabilities (e.g., with thin clients or pruning methods). Finally, we are currently working on transferring the system to a swarm of e-puck robots.

References

[1] M. Atzori. 2016. Blockchain-Based Architectures for the Internet of Things: A Survey. (2016). `https://ssrn.com/abstract=2846810`

[2] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo. 2013. Swarm robotics: A Review from the Swarm Engineering Perspective. *Swarm Intelligence* 7, 1 (2013), 1–41. `https://doi.org/10.1007/s11721-012-0075-2`

[3] A. Brutschy, G. Pini, C. Pinciroli, M. Birattari, and M. Dorigo. 2014. Self-organized Task Allocation to Sequentially Interdependent Tasks in Swarm Robotics. *Autonomous Agents and Multi-Agent Systems* 28, 1 (2014), 101–125. `https://doi.org/`

[4] V. Buterin. 2014. A next-generation smart contract and de-

centralized application platform. Ethereum Project White Paper. (2014). https://github.com/ethereum/wiki/wiki/White-Paper

[5] K. Christidis and M. Devetsikiotis. 2016. Blockchains and Smart Contracts for the Internet of Things. *IEEE Access* 4 (2016), 2292–2303. https://doi.org/10.1109/ACCESS.2016.2566339

[6] A. Dorri, S. S. Kanhere, and R. Jurdak. 2016. Blockchain in Internet of Things: Challenges and Solutions. *pre-print* (2016). arXiv:1608.05187v1

[7] E. C. Ferrer. 2016. The blockchain: A new framework for robotic swarm systems. *pre-print* (2016). arXiv:1608.00695v3

[8] L. Garattoni, G. Francesca, A. Brutschy, C. Pinciroli, and M. Birattari. 2015. *Software infrastructure for E-puck (and TAM)*. Tech. Rep. 2015-004. IRIDIA, Université libre de Bruxelles.

[9] H. Hamann, T. Schmickl, H. Wörn, and K. Crailsheim. 2012. Analysis of emergent symmetry breaking in collective decision making. *Neural Computing and Applications* 21, 2 (2012), 207–218. https://doi.org/10.1007/s00521-010-0368-6

[10] F. Higgins, A. Tomlinson, and K. M. Martin. 2009. Survey on Security Challenges for Swarm Robotics. In *Proc. Fifth Int. Conf. Autonomic and Autonomous Systems*. IEEE Press, 307–312. https://doi.org/10.1109/ICAS.2009.62

[11] A. G. Millard, J. Timmis, and A. F. T. Winfield. 2014. Towards Exogenous Fault Detection in Swarm Robotic Systems. In *Towards Autonomous Robotic Systems - Proceedings of TAROS 2013 - 14th Annual Conference (Lecture Notes in Computer Science)*, Vol. 8069. Springer, 429–430. https://doi.org/10.1007/978-3-662-43645-5_44

[12] M. A. Montes de Oca, E. Ferrante, A. Scheidler, C. Pinciroli, M. Birattari, and M. Dorigo. 2011. Majority-Rule Opinion Dynamics with Differential Latency: A Mechanism for Self-Organized Collective Decision-Making. *Swarm Intelligence* 5, 3–4 (2011), 305–327. https://doi.org/10.1007/s11721-011-0062-z

[13] S. Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008). https://bitcoin.org/bitcoin.pdf

[14] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo. 2012. ARGoS: A Modular, Parallel, Multi-Engine Simulator for Multi-Robot Systems. *Swarm Intelligence* 6, 4 (2012), 271–295. https://doi.org/10.1007/s11721-012-0072-5

[15] G. Pini, A. Brutschy, M. Frison, A. Roli, M. Dorigo, and M. Birattari. 2011. Task Partitioning in Swarms of Robots: An Adaptive Method for Strategy Selection. *Swarm Intelligence* 5, 3–4 (2011), 283–304. https://doi.org/10.1007/s11721-011-0060-1

[16] A. Prorok and V. Kumar. 2016. A Macroscopic Privacy Model for Heterogeneous Robot Swarms. In *Swarm Intelligence – Proceedings of ANTS 2016 – Tenth International Conference*, Marco Dorigo, Mauro Birattari, Xiaodong Li, Manuel López-Ibáñez, Kazuhiro Ohkura, Carlo Pinciroli, and Thomas Stützle (Eds.). Springer, 15–27. https://doi.org/10.1007/978-3-319-44427-7_2

[17] A. Reina, M. Dorigo, and V. Trianni. 2014. Collective Decision Making in Distributed Systems Inspired by Honeybees Behaviour. In *Proceedings of 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014)*. Int. Foundation for Autonomous Agents and Multiagent Systems, 1421–1422.

[18] I. Sargeant and A. Tomlinson. 2016. Maliciously manipulating a robotic swarm. In *Proc. of ESCS'16 – The 14th Intern. Conf. on Embedded Systems, Cyber-physical Systems, & Applications*. CSREA Press, 122–128.

[19] G. Valentini, D. Brambilla, H. Hamann, and M. Dorigo. 2016. Collective Perception of Environmental Features in a Robot Swarm. In *Swarm Intelligence – Proceedings of ANTS 2016 – Tenth International Conference (Lecture Notes in Computer Science)*, Vol. 9882. Springer, 65–76. https://doi.org/chapter/10.1007/978-3-319-44427-7_6

[20] G. Valentini, E. Ferrante, and M. Dorigo. 2017. The Best-of-n Problem in Robot Swarms: Formalization, State of the Art, and Novel Perspectives. *Frontiers in Robotics and AI* 4 (2017), 9. https://doi.org/10.3389/frobt.2017.00009

[21] G. Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper. (2014). http://gavwood.com/paper.pdf

[22] I. Zikratov, O. Maslennikov, I. Lebedev, A. Ometov, and S. Andreev. 2016. Dynamic trust management framework for robotic multi-agent systems. In *Proc. of 12th Int. Conf. on Next Generation Teletraffic and Wired/Wireless Advanced Networking, NEW2AN, and the 5th Conf. on Internet of Things and Smart Spaces, ruSMART 2016*, Olga Galinina, Sergey Balandin, and Yevgeni Koucheryavy (Eds.). Springer, 339–348.