A Generic Framework for Byzantine-tolerant Consensus Achievement in Robot Swarms

Hanqing Zhao[1], Alexandre Pacheco[2], Volker Strobel[2], Andreagiovanni Reina[2], Xue Liu[1], Gregory Dudek[1] and Marco Dorigo[2]

***Abstract*— Recent studies show that some security features that blockchains grant to decentralized networks on the internet can be ported to swarm robotics. Although the integration of blockchain technology and swarm robotics shows great promise, thus far, research has been limited to proof-of-concept scenarios where the blockchain-based mechanisms are tailored to a particular swarm task and operating environment. In this study, we propose a generic framework based on a blockchain smart contract that enables robot swarms to achieve secure consensus in an arbitrary observation space. This means that our framework can be customized to fit different swarm robotics missions, while providing methods to identify and neutralize Byzantine robots, that is, robots which exhibit detrimental behaviours stemming from faults or malicious tampering.**

I. Introduction

The ability to communicate individual opinions and to reach a collective agreement regarding those opinions—i.e., achieve consensus—is a fundamental behaviour in various societies of animals. Oftentimes, the survival and thriving of an animal society depend on how efficiently it can achieve consensus on environmental observations made by its members, as well as on which actions are the most appropriate response to those observations [1]. For instance, herds of mammals need to agree on when to move towards pastures, or towards water sources; colonies of insects need to choose the best location for nesting; and some troops of primates, including societies of humans, have systems to collectively select their leaders.

As a discipline which originated from imitating the collective behaviour of living organisms, over the years swarm robotics research has intensively studied consensus achievement in a population of robots [2]. To achieve consensus in a way that preserves the desired properties of a robot swarm—decentralization and flexibility—is not a trivial task. Classical consensus studies in swarm robotics are usually dedicated methods for specific tasks (e.g., path or site selection [3], [4], coordinated motion [5], or selection of the best option from a discrete set of alternatives [6]), and thus lack the flexibility to be adapted to the general-case consensus achievement problem, that is, agree on any subset of elements in any type of observation space. Furthermore, despite robot swarms being designed with redundancy, and thus being considered resilient to faults that disable the entire robot [7], recent research has shown that current consensus methods in swarm robotics are vulnerable to various types of attacks, such as disruptions caused by robots that behave differently from what was intended (e.g., due to hardware faults, or to malicious tampering from an adversary agent [8]–[10]); and Sybil attacks, where a single agent creates multiple participation credentials to control the result of consensus. Once again, addressing these challenges while maintaining a decentralized robot swarm is not trivial, since robot swarms lack a central authority that can assign participation credentials to robots and monitor their behaviour.

The Byzantine Generals Problem is a classical thought problem in which agents try to reach a consensus in a decentralized way, even though some of the agents are unreliable. In a seminal paper, it was shown that the problem is unsolvable when the number of agents capable of coordinating malicious attacks is greater than one-third of the total population [11], [12]. Inspired by the Byzantine Generals, we use the term *Byzantine behaviours* to refer to a broad set of robot behaviours that are caused by faults or malicious tampering, are difficult to detect, and can potentially lead to incorrect consensus. A robot that exhibits Byzantine behaviours is called a *Byzantine robot*. Incorrect consensus can inhibit the swarm from completing a higher-order task (e.g., if the robots agree on an incorrect position of a target object and waste time and energy to plan and act around this position) or, in the worst cases, can lead to a catastrophic result (e.g., if the robots agree on a hazardous building site location). Hence, it is crucial to consider the impact of Byzantine behaviours on consensus achievement in robot swarms.

Advances in distributed database technology and computing protocols (i.e., blockchains and smart contracts [13], [14]) potentially provide the necessary tools for robot swarms to achieve global consensus in a decentralized and secure manner [15]. The authors of [16], [17] deployed a blockchain-based smart contract in a robot swarm that enabled the robots to agree on an estimated ratio between white and black tiles on the experimental floor, despite the presence of Byzantine robots. However, their solution is tailored to that specific scenario and does not provide the means to distinguish different types of Byzantine behaviour. Indeed, Byzantine behaviours that are caused by faulty hardware or software and those that are caused by malicious tampering are anomalies which differ in nature and severity [18]; thus, distinguishing between them is a critical challenge in

[1] School of Computer Science, McGill University, Montréal, Canada
[2] IRIDIA, Université libre de Bruxelles, Brussels, Belgium

*This work was supported by Brussels International and by Québec MRIF via the Bilateral Cooperation Québec–Brussels-Capital Region. V. Strobel, A. Reina and M. Dorigo acknowledge financial support from the Belgian F.R.S.-FNRS. H. Zhao and G. Dudek acknowledge support from the NSERC Canadian Robotics Network (NCRN).

developing a Byzantine-tolerant robot swarm.

In this paper, we propose a generic and flexible framework that, exploiting the programmability of smart contracts, is suitable for any swarm robotics mission where achieving consensus in a secure manner is necessary. More specifically, our framework uses blockchain technology and smart contracts to enable robot swarms to achieve secure consensus in an arbitrary observation space, which means that our framework could be deployed to different swarm robotics missions, while granting crucial security properties against Byzantine behaviours and Sybil attacks.

The framework is implemented using a blockchain smart contract composed of three modules: 1) a clustering algorithm, which groups the observations from robots into clusters and where the centre of each cluster represents a piece of new information to be considered for inclusion in consensus; 2) a weighted voting game (WVG) protocol [19], which determines the inclusion, or not, of new information in consensus; and 3) a reward mechanism, which redistributes crypto assets from robots that vote against the outcome of the WVG to those that vote accordingly with the WVG outcome.

The crypto-economic aspect of the smart contract and the properties of crypto assets (e.g., scarcity and fungibility) are crucial to the security and operation of the framework. First, the balance of crypto assets owned by faulty and malicious robots converges to different values—indeed, malicious robots that consistently input erroneous information tend to lose most of their crypto assets, while faulty robots can maintain or gain crypto assets depending on how severe their faults are (i.e., how often they trigger Byzantine behaviours). This provides an intuitive way to assess the nature and severity of the robots' Byzantine behaviours. Second, the balance of crypto assets can be used to enable Byzantine fault tolerance: robots use crypto assets as a participation credential, which protects the swarm from Sybil attacks and from the harmful impact of Byzantine robots. Indeed, malicious and faulty robots eventually own fewer crypto assets due to their behaviours, thus becoming less impactful in the consensus achievement process [16].

We validate our framework using an experimental scenario in which the robot swarm must locate a set of resource patches (named food sources) that are scattered in the environment at unknown positions in order to subsequently perform collective resource collection (a foraging task) [20]. In this scenario, the goal of the swarm is to achieve consensus on a set of food source coordinates. The robots estimate their coordinates in a global reference frame, but no robot is able to make perfect measurements. The quality of their position estimates depends on the precision of their hardware and sensing; therefore, every robot has a different level of accuracy in making position estimates. In addition, malicious robots deliberately report incorrect food source coordinates, in an attempt to misguide the other robots.

This paper gives three major contributions:

1) A generic Byzantine-tolerant consensus framework that can be deployed to different robot swarm consensus tasks, particularly when security is a critical concern. This is accomplished by exploiting the programmability of smart contracts and the cryptographic security of blockchains.
2) A method to identify and neutralize Byzantine robots that can be executed during the autonomous operation of the swarm, i.e., without the need for a centralized authority to diagnose Byzantine behaviours nor to grant participation credentials to the robots. This is accomplished through the use of scarce crypto assets which serve both as a participation credential and as a metric to qualify Byzantine behaviours.
3) A solution, using the proposed framework, to a problem inspired by a swarm robotics foraging task, which allows the robot swarm (composed of robots with both faulty and malicious behaviours) to securely achieve consensus on sets of coordinates of food sources.

II. Related Work

Our approach is related to several topics as listed below.

a) Byzantine behaviour detection and identification: A key characteristic of Byzantine behaviours is the difficulty to establish whether or not they have occurred [12]. While some faults are straightforward and can be easily detected (e.g., disabled components), most faults that lead to Byzantine behaviours are caused by intermittent or partial faults which introduce deviations from the robot's intended and actual behaviours (e.g., miscalibrated sensors), and thus require a dedicated detection effort [21]. A robot with faults which lead to Byzantine behaviours may still contribute to the swarm operations, but its unreliability can pose a security risk [22], particularly in consensus achievement. Considering this, we classify Byzantine behaviours in two categories:

1) **Faulty behaviours**, which are caused by hardware or software faults and lead to unexpected changes in the functioning of the robot while they perform the normal (expected) control strategy [23].

2) **Malicious behaviours**, which are ascribed to the tampering of the robot control software or communications. The malicious robot executes strategies which are different from the expected strategy, as the result of an attack from an external agent with the objective of disrupting the swarm.

Faults that trigger Byzantine behaviours can be detected through either endogenous detection (when robots detect faults in themselves) or exogenous detection (when robot faults are detected by an external entity or another robot) [21], but malicious tampering can only be detected through exogenous detection. Furthermore, some approaches in exogenous Byzantine behaviour detection trust a centralized detection algorithm [24], [25] that is assumed to never fail, although this is unrealistic in practice.

In the absence of a centralized fault detector, Christensen et al. [26] designed a dedicated communication protocol to identify faulty robots, while Tarapore et al. [27] used an adaptive artificial immune system as a decentralized Byzantine behaviour detection protocol. However, these approaches require robots to perfectly follow a predefined behavioural protocol and/or correctly report diagnosis information based

on their internal state. Thus they are not applicable to unexpected faults, and will not detect malicious behaviours.

Our framework can be seen as an exogenous and fully decentralized approach to detect Byzantine behaviours in robot swarms while providing the ability to distinguish between Byzantine behaviours induced by faults from those induced by malicious tampering.

b) Consensus achievement in robot swarms: To solve a consensus achievement problem, the swarm has to agree on a choice among a set of choices. A distinction has been made between discrete and continuous consensus achievement problems. In the discrete case, robots need to agree on a finite set of possible choices (such as path or site selection [3], [6]), while in the continuous case, the robots need to agree on the values of a set of continuous variables (such as collective motion [5], or spatial aggregation [28]). Consensus achievement can be further divided into exact and approximate forms [29]. Exact consensus requires that every robot selects precisely the same choice, while approximate consensus allows robots to make different choices, as long as the difference between the maximum and the minimum values of these choices is small enough. Our approach is designed for exact consensus achievement and can be implemented in both discrete and continuous domains.

c) Blockchain for Byzantine-tolerant consensus achievement: The recent success of Decentralized Autonomous Organizations (DAOs) suggests that voting mechanisms can be used as Byzantine-tolerant consensus protocols in decentralized and open participation networks [30] by using crypto-economic incentives to indirectly regulate the behaviour of participants. DAOs exploiting the programming flexibility of blockchain smart contracts have been developed for various purposes, such as hosting stable coins and peer-to-peer exchanges, as well as enabling distributed decision-making within the DAO [31]–[33]. Strobel et al. [16] first introduced blockchain smart contracts as a distributed platform to achieve secure consensus in robot swarms, while Pacheco et al. [17] showed the viability of deploying this platform on a swarm composed of real Pi-puck robots. They proposed a blockchain smart contract as an algorithmic way to achieve secure consensus on one-dimensional sensor readings. However, their study was specific to a particular task and observation space and did not include Byzantine identification to differentiate between faults or malicious behaviours. This paper extends this line of research by designing a generic blockchain-based consensus framework (i.e., a DAO) for swarm robotics applications, in which a voting-based mechanism enables Byzantine-tolerant consensus achievement.

III. Generic Framework

We consider the Byzantine-tolerant consensus achievement problem in an observation space Ω, where a swarm $\mathbb{S}$ has to agree on a *consensus set* L in the power set of the observation space $pow(\Omega)$. For example, if the robots' observations are coordinates (i.e., points on a 2D space), then $\Omega = \mathbb{R}^2$, $pow(\Omega)$ represents any possible set of points $\mathbb{R}^2$,

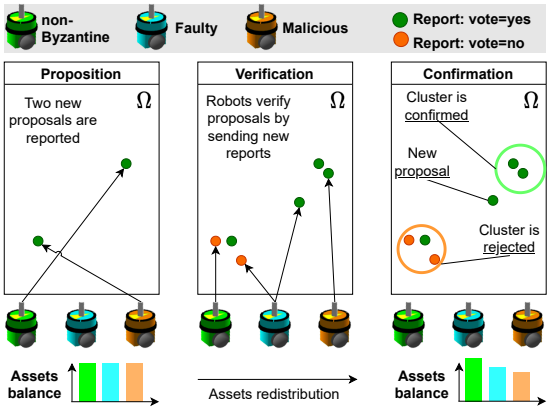

Fig. 1. Illustration of the three major stages of our framework given a swarm of three robots with different behaviours. The non-Byzantine robot performs the normal strategy with good quality observations; the faulty robot behaves as the non-Byzantine but with high observation error; and the malicious robot reports incorrect observations. In the Proposition stage, robots submit reports that do not fall within existing clusters, thus forming new proposals. In the Verification stage, robots verify proposals by submitting additional reports, thus updating the clusters and improving the food source position estimate. Reports that do not fall within existing clusters, become new proposals (even if sent during the Verification stage). In the Confirmation stage, clusters that are sufficiently verified are either confirmed or rejected according to a voting process. Throughout this process, the assets of the robots are redistributed according to their contributions.

and the goal of the consensus achievement problem is to let all robots agree on a particular set of points $L \in pow(\Omega)$.

Our framework to achieve secure consensus can be divided into three main stages, which are visualized in Fig. 1.

a) ***Proposition****:* Robots submit *reports* to the smart contract (see Definition 1). Each report includes an observation in the Ω space that the robot believes should be added to the consensus set. A report that is not similar enough to previous reports is classified by the smart contract as a *proposal*, which must be verified by the other robots.

b) ***Verification****:* Robots consult the smart contract to get the list of unverified proposals (i.e., proposals that have not been added to the consensus set yet). Verification may require an action from the robots, such as sensing the environment or performing a computation. Afterwards, the robot submits a *report* of the observations that result from this action. Once a proposal has received enough reports, it can enter the confirmation stage.

c) ***Confirmation****:* The smart contract confirms or not the membership to the consensus set of the *proposal*, and rewards crypto assets to the robots that participated in the proposition and verification of the confirmed proposal. At this stage, the consensus set is permanently changed on the blockchain and, once all robots synchronize their blockchains, a new consensus has been achieved.

A report is a structured blockchain transaction defined as follows.

Definition 1. A *report* is a 4-tuple $\boldsymbol{r} = (o, v, w, x)$, which consists of an observation $\boldsymbol{r}.o \in \Omega$, a voting decision $\boldsymbol{r}.v \in \{yes, no\}$ representing the robots' opinion on whether or not the observation $\boldsymbol{r}.o$ should become part of the consensus

set, a deposit of an amount $\boldsymbol{r}.w$ of crypto assets to the smart contract address, and the blockchain address $\boldsymbol{r}.x$ of the reporting robot.

The smart contract uses a clustering algorithm to create clusters and assign each report received during the Proposition and Verification stages to a *cluster* based on its similarity to other reports. The similarity between reports is a distance measure on the observation space $d : \Omega \times \Omega \to \mathbb{R}_{\geq 0}$.

Definition 2. A *cluster* $C_i \in \mathcal{C}$ is a set of reports $\boldsymbol{r}$ such that $\forall \boldsymbol{r} \in C_i : d(\boldsymbol{r}.o, m(C_i)) \leq d(\boldsymbol{r}.o, m(C_j)), \forall C_j \in \mathcal{C} \backslash C_i$, where $\mathcal{C}$ is the set of all clusters, and $m(C_i)$ represents the centre of the cluster C_i. The cluster centres are initialized according to the rules employed by the clustering algorithm, and then updated at each iteration as the weighted average of the reports in C_i, i.e., $m(C_i) = \sum_{\boldsymbol{r} \in C_i} \boldsymbol{r}.o \cdot \boldsymbol{r}.w / |C_i|$.

During the Verification stage, robots perform actions to verify clusters. A cluster C_i is *sufficiently verified* if and only if its reports satisfy the following conditions:

1) *Unique identities:* $\forall (\boldsymbol{r}_a, \boldsymbol{r}_b) \in C_i^2 : \boldsymbol{r}_a.x \neq \boldsymbol{r}_b.x,\ a \neq b$
2) *Assets supermajority:* $\sum_{\boldsymbol{r} \in C_i} \boldsymbol{r}.w \geq \frac{2}{3} q_r T,$

where T is the total supply of crypto assets in circulation, and $q_r \in [0, 1]$ is the *Relative assets quota*, a parameter that defines the reporting deposit as a percentage of robots' current balance [34]. The value of q_r controls the maximum number of unconfirmed clusters on the smart contract. In practice, a higher value of q_r will accelerate the process of the redistribution of assets between robots, but will also slow down the process of adding new observations to the consensus set (in our experiments, we use $q_r = \frac{1}{3}$).

As a prerequisite for Byzantine tolerance, it is assumed that malicious robots will never be able to carry out a coordinated attack with control of over one-third of the total assets. If such control were to happen, coordinated attackers could potentially submit false reports with identical information or refuse to send reports, thus misleading or obstructing the achievement of consensus, respectively.

Definition 3. A *proposal* is the centre of a cluster C_i that is currently in the Verification stage.

Once a cluster C_i has been sufficiently verified, the corresponding proposal enters the Confirmation stage for drawing a collective decision on whether or not it belongs in the consensus set. For this purpose, a Weighted Voting Game (WVG) is conducted over all reports in C_i.

The WVG weight vector consists of the crypto assets deposited for each report in C_i: $\boldsymbol{w} = (\boldsymbol{r}_1.w, ..., \boldsymbol{r}_{|C_i|}.w) \in \mathbb{R}_{\geq 0}^{|C_i|}$. The reports in C_i are divided into two coalitions Y_{C_i} and N_{C_i} according to their votes: $Y_{C_i} = \{\boldsymbol{r} \in C_i | \boldsymbol{r}.v = yes\}$ and $N_{C_i} = C_i \backslash Y_{C_i}$. A coalition $W_{C_i} \in \{Y_{C_i}, N_{C_i}\}$ is the winning coalition if it satisfies a *voting rule*; in our experiment, we use the weighted majority rule: W_i is the winning coalition if $\sum_{\boldsymbol{r} \in W_{C_i}} \boldsymbol{r}.w \geq \frac{1}{2} \sum_{\boldsymbol{r} \in C_i} \boldsymbol{r}.w$.

When Y_{C_i} becomes the winning coalition of the WVG, the proposal of C_i is confirmed as a member of the consensus set. After the WVG is concluded, all reports in C_i are removed from the smart contract state, and the deposited crypto assets in these reports are redistributed according to a *reward function*.

Definition 4. A *reward function* f takes a sufficiently verified cluster C_i, the WVG winning coalition W_{C_i}, and a report $\boldsymbol{r} \in C_i$. It returns the amount of crypto assets that need to be transferred to the reporter robot address $\boldsymbol{r}.x$.

The reward function can have different forms depending on the tasks and observation spaces. In our experiment, we used the following reward function:

$$f(\boldsymbol{r}, C_i, W_{C_i}) = \begin{cases} \boldsymbol{r}.w + \frac{\sum_{\boldsymbol{r}' \in C_i \backslash W_{C_i}} \boldsymbol{r}'.w}{|W_{C_i}|}, & \text{if } \boldsymbol{r} \in W_{C_i} \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

where $C_i \backslash W_{C_i}$ represents the losing coalition in the WVG. In this reward function, all robots in the winning coalition equally share the deposits from the robots in the losing coalition (second term in Eq. 1, top). Otherwise, using a weight-based redistribution that is proportional to the amount of deposits may lead to a rich-get-richer phenomenon and thus centralization of the framework.

After the confirmation of a cluster C_i, the weighted average $m(W_{C_i})$ of all observations $\boldsymbol{r}.o$ in the *supportive* reports $\boldsymbol{r} \in W_{C_i}$ within the cluster will be included in the swarm consensus set:

$$m(W_{C_i}) = \frac{\sum_{\boldsymbol{r} \in W_{C_i}} \boldsymbol{r}.o \cdot \boldsymbol{r}.w}{|W_{C_i}|}. \tag{2}$$

IV. Experimental Study

A. Problem statement and the smart contract

We apply our framework to a food source location discovery task, where the observations are 2D coordinates in a global reference frame, that is, $\Omega = \mathbb{R}^2$. The swarm has to reach a consensus over a set of available food source coordinates $L \in pow(\Omega)$. The smart contract meta controller in our experiments is shown in Algorithm 1.

The meta controller includes two executable functions. `Report` is a function which enables robots to submit their reports to the consensus algorithm accompanied by the transaction of the crypto assets deposit. `ConsultProposals` is a call-only function which returns centres of clusters that are awaiting verification (i.e., proposals), in order to coordinate the robots' verification actions. Note that the smart contract does not distinguish the reports made during the Proposition or Verification stages, as it is the outcome of the clustering algorithm that defines whether a report is a new proposal or a verification of a current one.

An incremental k-means algorithm [35], implemented on the meta controller, is responsible for clustering similar reports and identifying outliers, which become new proposals. Note that although there exist more complex search strategies for k value adjustment in k-means algorithms [36], in this paper, we use a simple approach to adjust the k value: we increase k by one when the distance between a new observation that is greater than a threshold τ from all existing cluster centres is reported.

Algorithm 1: Smart contract meta controller

Input : Swarm $\mathbb{S}$; Observation space Ω; Distance function d; Inter-cluster distance threshold τ; Relative assets quota q_r; Total circulating assets T
Output: Consensus set $L \in pow(\Omega)$
Init : Size of consensus set $k = 0$; Cluster set $C = \{\}$; Report set $R = \{\}$; Consensus set $L = \{\}$.

Procedure `Report` $(\boldsymbol{r} = (o, v, w, x))$
- **if** $w < q_r$*Balance*$(\boldsymbol{r}.x)$ **then**
  - ▷ `Reject for insufficient deposit`
  - Transfer $\boldsymbol{r}.w$ assets to $\boldsymbol{r}.x$; return 0;
- $R \leftarrow R \bigcup \{\boldsymbol{r}\}$;
- **if** $\mathcal{C} = \emptyset$ **then**
  - $\mathcal{C} \leftarrow \mathcal{C} \bigcup \{\boldsymbol{r}\}$; $k \leftarrow 1$;
- **else**
  - ▷ `Unsupervised clustering`
  - **if** $\forall C_i \in \mathcal{C} : d(\texttt{Centre}(C_i), r) \geq \tau \wedge k < \lfloor \frac{1}{q_r} \rfloor$ **then**
    - $k \leftarrow k + 1$
  - $\mathcal{C} \leftarrow \texttt{IncreKmeans}(R, k)$;
- **for** $C_i \in \mathcal{C}$ **do**
  - **if** $\exists (\boldsymbol{r}_1, \boldsymbol{r}_2) \in C_i^2 : \boldsymbol{r}_1.x = \boldsymbol{r}_2.x$ **then**
    - ▷ `Remove duplicate identities`
    - $R \leftarrow R \backslash \{\boldsymbol{r}_2\}; C_i \leftarrow C_i \backslash \{\boldsymbol{r}_2\}$;
    - Transfer $\boldsymbol{r}_2.w$ assets to $\boldsymbol{r}_2.x$;
  - **if** $\sum_{\boldsymbol{r} \in C_i} \boldsymbol{r}.w \geq \frac{2}{3} q_r W$ **then**
    - ▷ `Perform WVG over all` $\boldsymbol{r} \in C_i$
    - **if** Y_{C_i} *is winning coalition* **then**
      - ▷ `Most assets vote yes`
      - $L \leftarrow L \bigcup \texttt{Centre}(C_i)$;
    - **for** $\boldsymbol{r}' \in$ *winning coalition* **do**
      - Transfer $f(\boldsymbol{r}', C_i)$ assets to $\boldsymbol{r}'.x$;
    - $\mathcal{C} \leftarrow \mathcal{C} \backslash \{C_i\}$; $R \leftarrow R \backslash C_i$; $k \leftarrow k - 1$;

Procedure `ConsultProposals()`
- **return** $\{\texttt{Centre}(C_i) : \forall C_i \in \mathcal{C}, \sum_{\boldsymbol{r} \in C_i} \boldsymbol{r}.w < \frac{2}{3} q_r T\}$;

B. Simulation setup

We simulate the food source discovery task in a square-shaped arena in the ARGoS simulator [37], [38], where the food sources and a swarm of $n = |\mathbb{S}|$ robots are initially placed at random positions in the arena (Fig. 2). Each robot runs an Ethereum Virtual Machine (EVM) instance, associated with a unique wallet address, that handles the distributed database and executes the computations of the smart contract for consensus achievement, which is implemented in the Solidity programming language. To guarantee consensus on the distributed database, the swarm uses the Proof-of-Authority (PoA) consensus protocol [39]. Compared to the classical Proof-of-Work consensus, PoA does not spend the computational resources of robots to solve mathematical puzzles and has a predictable new block generation period. When participants are perfectly time-synchronized, PoA is Byzantine-tolerant up to $\frac{n}{2} - 1$ Byzantine participants [40]. However, if this is not the case, this value drops to the theoretical limit of $\frac{n}{3}$ [41], [42]. During our swarm robotics experiments, synchronization delays can occur due to sparse connectivity between the robots operating in a large environment. For this reason, we use a block generation period of 2 seconds, to let the information blocks have enough time to be synchronized. Additionally, we disregard the impact of gas fees (fees paid in crypto assets for the execution of smart contract functions) on the operation of smart contracts. The block generation period can be thought of as the operating frequency of the smart contract meta controller [20]. For more details on our simulations setup, we refer to [43].

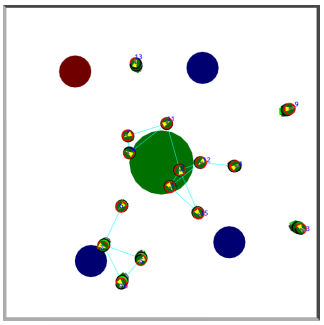

Fig. 2. Top view of the square arena (side length: $l_a = 1.5\,\text{m}$) with 15 robots. The arena contains three food sources (blue). Robots only get noise-free positioning within the home region (green). The red circle indicates the incorrect food source reported by the malicious robot, which is not visible to the other robots. Robots connected with cyan lines are within communication range and can synchronize their blockchains.

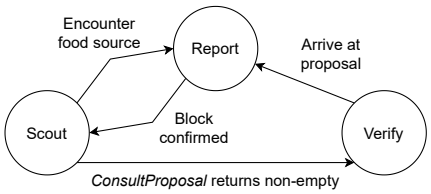

Fig. 3. The FSM controller of a non-malicious robot.

C. Robot controller

All non-malicious robots are equipped with a three-state Finite State Machine (FSM) controller shown in Fig. 3. The robot behaviour in each state of the FSM is the following:

1) **Scout:** Move randomly in the arena while sensing the floor colour using the ground sensors and querying the `ConsultProposals` function on the smart contract at regular intervals. Then, transition to the *Verify* state when `ConsultProposals` returns a non-empty result or to the *Report* state when the ground sensors detect blue colour.
2) **Verify:** Randomly select a proposal from the list of proposals from `ConsultProposals`, then navigate to the proposal coordinates to perform verification. Afterwards, transition to the *Report* state.
3) **Report:** Construct a report with the estimates of the centre of the food source measured with the ground sensors, then send a transaction containing the report and corresponding deposit using the `Report` function on the smart contract. Afterwards, switch to *Scout* state once the report has been added to the blockchain.

D. Byzantine behaviours

Among the swarm of n robots, we assume that any robot may exhibit Byzantine behaviour, either due to faults with

varying degrees of severity or malicious tampering. More precisely, n_f robots are faulty robots—each one of them has different and imperfect self-positioning quality—while n_m robots are malicious and might try to mislead the swarm by reporting incorrect food source positions. The number of robots satisfies $n_f + n_m = n$.

Each **faulty robot** is indexed by an integer $n_i = 1 \dots n_f$: robots with higher indexes have more severe faults, and thus, higher observation error. We simulate two different types of faults that lead to inaccurate observations of the 2D coordinates of food sources:

a) Mechanical faults: We simulate friction on wheels, which makes the actual speed of the wheel lower than the command speed. The speed decrease on each wheel at each time step is sampled from a uniform distribution between $[0, p_m n_i]$, where $p_m = 0.01$ stands for the unit friction parameter and n_i represents the index of the robot.

b) Sensing faults: The robot estimates its current position using readings from a noisy virtual sensor. At each time step t, the sensor provides a position estimate $\boldsymbol{y}_t = \boldsymbol{x}_t + \boldsymbol{z}_t$, where $\boldsymbol{x}_t$ is the actual position of the robot and $\boldsymbol{z}_t \sim \mathcal{N}(0, V)$ is Gaussian noise, with $V = I p_s n_i$ a covariance matrix where I is the identity matrix and $p_s = 0.03$. This virtual sensor mimics a self-localization error from reference landmarks, usually generated by more complex sensors and algorithms (e.g., camera, IMU and SLAM). The reading from the virtual sensor is noise-free when robots are located in the home region.

To build a new report over the observation space, each robot predicts its position in the global reference frame using a Kalman filter [44] with ideal forward dynamics, of which the state space is equal to the observation space $\Omega = \mathbb{R}^2$ (positions in 2D). In the prediction step, the robot predicts its position by applying wheel speed action to the forward dynamics model in the Kalman filter, while in the update step, the robot observes a noisy position from the virtual sensor and updates its position estimate.

The **malicious robots** follow a similar FSM controller to the one shown in Fig. 3, but instead of performing random-walk during the *Scout* state, they move towards an incorrect food source location (represented by a red circle in Fig. 2), and upon arrival at the location, they submit a misleading report $\boldsymbol{r} : \boldsymbol{r}.v = yes$ supporting that incorrect source.

E. Comparative Metrics and Results

We compare our approach with two baseline consensus algorithms, namely, the linear consensus (LCP) [29] and weighted-mean-subsequence reduced (W-MSR) protocols [45]. In both LCP and W-MSR, a robot updates its belief according to the beliefs received from its neighbours, however, with W-MSR the robot goes a step further by discarding five beliefs that have the greatest Euclidean distance from its own belief.

Although both LCP and W-MSR are designed for achieving decentralized consensus, they differ from our approach in two critical ways. First, LCP and W-MSR are not applicable to the scenario with flexible cardinality of the consensus set (i.e., variable number of food source locations). Second, they are designed to achieve *approximate* consensus (i.e., each robot may hold slightly different beliefs), while in exact consensus protocols, such as ours, all robots are required to converge to the same belief. For these reasons, to draw a comparison with the LCP and W-MSR baselines, we conduct experiments with a single food source location and we define the *average consensus error* (E_c):

$$E_c = \sum_{a \in \mathbb{S}} \frac{||\boldsymbol{b}_a - \boldsymbol{p}_f||}{|\mathbb{S}|}, \tag{3}$$

where $\boldsymbol{b}_a$ is the belief of robot a and $\boldsymbol{p}_f$ is the correct position of the food source. With our exact consensus method, all robots share an identical belief denoted by $\boldsymbol{b}_a$, which corresponds to the consensus set obtained after the first confirmation of a cluster in each experiment (Eq. 2). As such, the consensus error becomes $E_c = ||\boldsymbol{b}_a - \boldsymbol{p}_f||$ (the same for all robots). Conversely, in the approximate consensus protocols, since each robot has its own belief, E_c must be calculated *post hoc* as the average of the consensus errors of the faulty robots (we excluded the malicious robots from $\mathbb{S}$ in Eq. 3, as, when using the LCP or W-MSR protocols, they never update their erroneous $\boldsymbol{b}_a$).

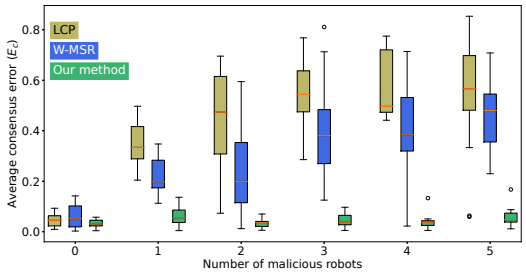

Fig. 4. Consensus error for increasing number of malicious robots in a swarm of 15 robots over 15 repetitions in a single food source arena. Compared to the baselines, our approach achieves lower consensus errors under the presence of 1 to 5 malicious robot(s).

Fig.4 shows that, unlike the baseline approaches, the consensus quality of our approach is not affected by the attack from malicious robots. However, it is important to weigh the trade-off between security and time and energy costs. For this purpose, we refer to the concept of *sensing cost* [46]. Specifically, we examine the total number of times the robots enter the *Report* state before the swarm achieves consensus. The transition conditions and behaviours of the *Report* state for our approach are defined in Fig. 3.

For the LCP and W-MSR baselines, the robot controller switches to the *Report* state when it detects a food source. However, if the robot is unable to find the food source but has received beliefs from at least two-thirds of robots in the swarm, it will take the average of all received beliefs as its initial belief, and will not switch to the *Report* state. As a result, with a swarm of 15 robots, the sensing costs of the LCP and W-MSR baseline methods are always between 10 and 15 (Fig. 5). Our approach, on the other hand, requires robots to repeatedly estimate the centre of food sources and

report their estimates to validate correct proposals and reject false ones, resulting in a higher sensing cost. This trade-off is critical in achieving Byzantine-tolerant consensus.

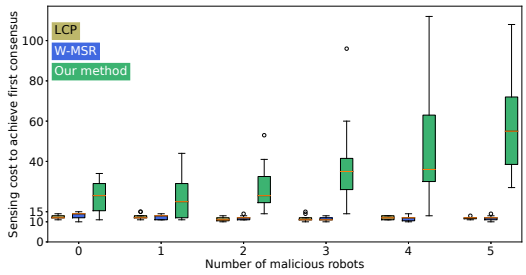

Fig. 5. Sensing cost for increasing number of malicious robots in a swarm of 15 robots over 15 repetitions in a single food source arena. Our approach requires higher sensing cost compared to the baselines.

One of the main features of our consensus framework is its flexibility in accommodating consensus sets with arbitrary cardinality. To demonstrate this, we further evaluate our approach in a more complex environment with three food sources as shown in Fig. 2. In this set of experiments, we consider a swarm of $n = 15$ robots, in which $n_f = 14$ are faulty robots and $n_m = 1$ robot is malicious. The faulty robots are indexed from 1 to 14 and show increasing severity of faults, as described in Section IV-D. To study the impact of asset redistribution within the swarm, we remove food source coordinates from the consensus set as soon as they are confirmed, allowing the robots to continue exploring and achieving new consensus.

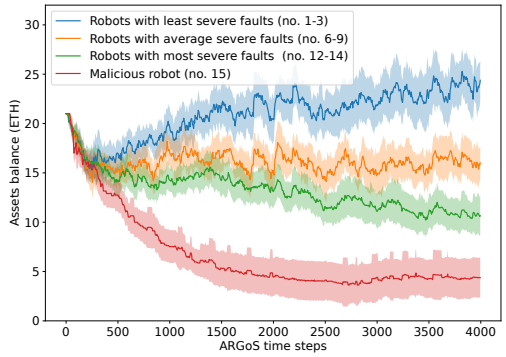

Fig. 6. Assets balance over time for a swarm of 15 robots over 15 repetitions. The averaged balances of robots with the least, average, and most severe faults are represented by the blue, orange, and green lines, respectively. The balance of the malicious robot is depicted in red, and the filled range represents the 95% confidence interval under a two-sided t-test. Parameter values are: $p_m = 0.01, p_s = 3l_a \times 10^{-4}$, where $l_a = 1.5\,\text{m}$ is the length of the arena.

As we can see from Fig. 6, the smart contract meta controller redistributes the assets among robots, according to their contribution to the consensus achievement. Robots with less severe faults are able to collect more assets and therefore have a higher "right to speak", while the malicious robot gradually loses all of its assets and is eventually excluded from participating in consensus achievement. Moreover, the distinct pattern of balance evolution for robots with different kinds and severity of Byzantine behaviours provides a metric which can be used to identify Byzantine robots.

To study how the tolerance to Byzantine behaviours evolves during an experiment, we define the *relative correct deposit (RCD)* as the fraction of crypto assets that are deposited in favour of existing food sources, i.e., those that are not wrongfully introduced by faulty or malicious robots:

$$RCD(C_i) = \frac{\sum_{\boldsymbol{r} \in Y_{C_i}} \boldsymbol{r}.w}{\sum_{\boldsymbol{r} \in C_i} \boldsymbol{r}.w}, \tag{4}$$

where C_i denotes a confirmed cluster, the centre of which lies within the area of an existing food source (see Fig. 2).

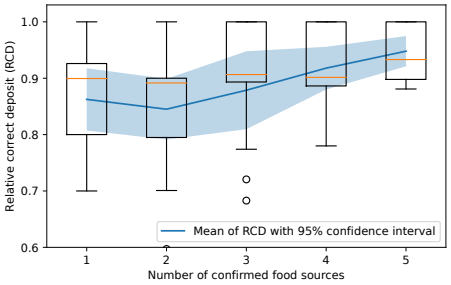

Fig. 7. RCD evolution on first 5 confirmations of existing food sources over 20 repetitions. The shaded area represents the 95% confidence interval under a two-sided t-test. The RCD evolution indicates a statistically increasing confidence of the swarm to include correct food source positions to the consensus set by giving more weight to robots with less severe faults, than to those presenting either severe faults or malicious behaviour.

Each time a new food source is confirmed (x-axis in Fig. 7), a redistribution of crypto assets occurs. This results in an increasing trend of the RCD since, after each new confirmation, assets are re-allocated to better-performing robots (Fig. 6), thus increasing the level of tolerance against incorrect reports, and reducing the influence that Byzantine behaviours can have on consensus achievement.

V. Conclusion

We have proposed a generic framework which uses blockchain and smart contracts to achieve consensus in a robot swarm in the presence of Byzantine robots. To prove the effectiveness of the framework, we implemented it as a meta controller using an Ethereum smart contract and compared it with the existing LCP and W-MSR protocols as baselines in a swarm robotics foraging scenario. Results show that our approach achieves better Byzantine tolerance, albeit at a higher sensing cost. Upon further examination of a more complex foraging scenario, we observed that our approach effectively redistributes crypto assets between robots based on the type and severity of faults exhibited by the robots. This redistribution led to increased robustness against Byzantine behaviours. Additionally, the amount of assets can serve as a metric to assess the severity of faults and to distinguish between faulty and malicious robots.

The generic nature of our framework is demonstrated through its flexibility and modularity: in practice, and as

long as robots provide observations that belong in the same observation space, no modifications are required in order to deploy our framework to a different consensus achievement scenario. In the case that a task requires a different clustering scheme for its observations, or different Byzantine-tolerance properties, the framework's modular design allows the clustering algorithm, the voting rule and the reward function to be tailored to the new task.

REFERENCES

[1] T. Kameda, T. Wisdom, W. Toyokawa, and K. Inukai, "Is consensus-seeking unique to humans? A selective review of animal group decision-making and its implications for (human) social psychology," *Group Proc. & Intergroup Relations*, vol. 15, no. 5, pp. 673–689, 2012.

[2] M. Dorigo, G. Theraulaz, and V. Trianni, "Swarm robotics: Past, present and future," *Proceedings of the IEEE*, vol. 109, no. 7, pp. 1152–1165, 2021.

[3] A. Campo, Á. Gutiérrez, S. Nouyan, C. Pinciroli, V. Longchamp, S. Garnier, and M. Dorigo, "Artificial pheromone for path selection by a foraging swarm of robots," *Biological Cybernetics*, vol. 103, no. 5, pp. 339–352, 2010.

[4] M. A. Hsieh, Á. Halász, S. Berman, and V. Kumar, "Biologically inspired redistribution of a swarm of robots among multiple sites," *Swarm Intelligence*, vol. 2, no. 2, pp. 121–141, 2008.

[5] E. Ferrante, A. E. Turgut, C. Huepe, A. Stranieri, C. Pinciroli, and M. Dorigo, "Self-organized flocking with a mobile robot swarm: A novel motion control method," *Adaptive Behavior*, vol. 20, no. 6, pp. 460–477, 2012.

[6] G. Valentini, E. Ferrante, and M. Dorigo, "The best-of-n problem in robot swarms: Formalization, state of the art, and novel perspectives," *Frontiers in Robotics and AI*, vol. 4, p. 9, 2017.

[7] M. Dorigo, M. Birattari, and M. Brambilla, "Swarm robotics," *Scholarpedia*, vol. 9, no. 1, p. 1463, 2014.

[8] V. Strobel, E. Castelló Ferrer, and M. Dorigo, "Managing Byzantine robots via blockchain technology in a swarm robotics collective decision making scenario," in *Proc. of AAMAS*, 2018, pp. 541—549.

[9] G. De Masi, J. Prasetyo, R. Zakir, N. Mankovskii, E. Ferrante, and E. Tuci, "Robot swarm democracy: The importance of informed individuals against zealots," *Swarm Intelligence*, vol. 15, no. 4, pp. 315–338, 2021.

[10] V. Strobel, A. Pacheco, and M. Dorigo, "Robot swarms neutralize harmful byzantine robots using a blockchain-based token economy," *Science Robotics*, vol. 8, no. 79, p. eabm4636, 2023.

[11] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *J. of the ACM*, vol. 27, no. 2, pp. 228–234, 1980.

[12] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.

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

[14] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger." Ethereum Foundation, Tech. Rep., 2014.

[15] A. Reina, "Robot teams stay safe with blockchains," *Nature Machine Intelligence*, vol. 2, pp. 240–241, 2020.

[16] V. Strobel, E. Castelló Ferrer, and M. Dorigo, "Blockchain technology secures robot swarms: A comparison of consensus protocols and their resilience to Byzantine robots," *Frontiers in Robotics and AI*, vol. 7, p. 54, 2020.

[17] A. Pacheco, V. Strobel, and M. Dorigo, "A blockchain-controlled physical robot swarm communicating via an ad-hoc network," in *Swarm Intelligence – Proceedings of ANTS*, ser. LNCS, vol. 12421. Springer, 2020, pp. 3–15.

[18] Y. Qian, D. Tipper, P. Krishnamurthy, and J. Joshi, *Information Assurance: Dependability and Security in Networked Systems*. Burlington, MA, USA: Elsevier/Morgan Kaufmann, 2010.

[19] F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. D. Procaccia, *Handbook of Computational Social Choice*. New York, NY, USA: Cambridge University Press, 2016.

[20] A. Pacheco, V. Strobel, A. Reina, and M. Dorigo, "Real-time coordination of a foraging robot swarm using blockchain smart contracts," in *Swarm Intelligence – Proceedings of ANTS*, ser. LNCS, vol. 13491. Springer, 2022, pp. 196–208.

[21] A. L. Christensen, "Fault detection in autonomous robots," Ph.D. dissertation, Université Libre de Bruxelles, 2008.

[22] E. R. Hunt and S. Hauert, "A checklist for safe robot swarms," *Nature Machine Intelligence*, vol. 2, no. 8, pp. 420–422, 2020.

[23] J. Chen and R. J. Patton, *Robust Model-Based Fault Diagnosis for Dynamic Systems*. New York, NY, USA: Springer, 1999, vol. 3.

[24] A. G. Millard, J. Timmis, and A. F. Winfield, "Run-time detection of faults in autonomous mobile robots based on the comparison of simulated and real robot behaviour," in *Proceedings of the IEEE IROS*. Piscataway, NJ, USA: IEEE Press, 2014, pp. 3720–3725.

[25] B. Khaldi, F. Harrou, F. Cherif, and Y. Sun, "Monitoring a robot swarm using a data-driven fault detection approach," *Robotics and Autonomous Systems*, vol. 97, pp. 193–203, 2017.

[26] A. L. Christensen, R. O'Grady, and M. Dorigo, "From fireflies to fault-tolerant swarms of robots," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 4, pp. 754–766, 2009.

[27] D. Tarapore, A. L. Christensen, and J. Timmis, "Generic, scalable and decentralized fault detection for robot swarms," *PLOS One*, vol. 12, no. 8, p. e0182058, 2017.

[28] O. Soysal and E. Sahin, "Probabilistic aggregation strategies in swarm robotic systems," in *Proceedings of the IEEE SIS*. Piscataway, NJ, USA: IEEE Press, 2005, pp. 325–332.

[29] J. Beal, "Trading accuracy for speed in approximate consensus," *The Knowledge Engineering Review*, vol. 31, no. 4, pp. 325–342, 2016.

[30] Y. El Faqir, J. Arroyo, and S. Hassan, "An overview of decentralized autonomous organizations on the blockchain," in *Proceedings of the OpenSym 2020*. New York, NY, USA: ACM Press, 2020, pp. 1–8.

[31] L. Hickey and M. Harrigan, "The Bisq DAO: On the privacy cost of participation," in *Proceedings of the ISCC*. IEEE, 2020, pp. 1–6.

[32] J. Potts, D. W. Allen, C. Berg, A. M. Lane, and T. MacDonald. (2022) The exchange theory of Web3 governance (or 'blockchains without romance'). [Online]. Available: https://ssrn.com/abstract=4209827

[33] M. Brennecke, T. Guggenberger, B. Schellinger, and N. Urbach, "The de-central bank in decentralized finance: A case study of MakerDAO," in *Proceedings of the HICSS*, 2022.

[34] I. Lindner, "The power of a collectivity to act in weighted voting games with many small voters," *Social Choice and Welfare*, vol. 30, no. 4, pp. 581–601, 2008.

[35] D. T. Pham, S. S. Dimov, and C. Nguyen, "An incremental k-means algorithm," *Proc.s of the Inst. of Mechanical Engineers, Part C: J. of Mechanical Engineering Science*, vol. 218, no. 7, pp. 783–795, 2004.

[36] P. Bholowalia and A. Kumar, "EBK-means: A clustering technique based on elbow method and k-means in WSN," *International Journal of Computer Applications*, vol. 105, no. 9, 2014.

[37] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, T. Stirling, Álvaro Gutiérrez, L. M. Gambardella, and M. Dorigo, "ARGoS: A modular, multi-engine simulator for heterogeneous swarm robotics," in *Proceedings of the IEEE IROS*, 2011, pp. 5027–5034.

[38] 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, "ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems," *Swarm Intelligence*, vol. 6, no. 4, pp. 271–295, 2012.

[39] S. De Angelis, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, "PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain," in *Proceedings of the ITASEC*, 2018.

[40] P. Szilágyi, "EIP 225: Clique proof-of-authority consensus protocol," 2017, https://github.com/ethereum/EIPs/issues/225.

[41] P. Ekparinya, V. Gramoli, and G. Jourjon, "The attack of the clones against proof-of-authority," in *Proceedings of the NDSS*, 2020.

[42] Q. Wang, R. Li, Q. Wang, S. Chen, and Y. Xiang, "Exploring unfairness on proof of authority: Order manipulation attacks and remedies," in *Proceedings of the ACM ASIACCS*, 2022, pp. 123–137.

[43] A. Pacheco, V. Strobel, and M. Dorigo, "A framework for swarm robotics experimentation with Pi-puck robots and an Ethereum-based blockchain," IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, Tech. Rep. TR/IRIDIA/2020-001, 2020.

[44] G. Welch and G. Bishop, "An introduction to the Kalman filter," University of North Carolina, Tech. Rep. TR 95-041, 1995.

[45] H. J. LeBlanc, H. Zhang, X. Koutsoukos, and S. Sundaram, "Resilient asymptotic consensus in robust networks," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 4, pp. 766–781, 2013.

[46] M. Tan, "Cost-sensitive learning of classification knowledge and its applications in robotics," *Machine Learning*, vol. 13, pp. 7–33, 1993.