Robot swarms neutralize harmful Byzantine robots using a blockchain-based token economy

Volker Strobel,[1*] Alexandre Pacheco,[1] Marco Dorigo[1]

[1]IRIDIA, Université Libre de Bruxelles, Brussels, Belgium

[*]To whom correspondence should be addressed; E-mail: volker.strobel@ulb.be.

Through cooperation, robot swarms can perform tasks or solve problems that a single robot from the swarm could not perform/solve by itself. However, it has been shown that a single Byzantine robot (such as a malfunctioning or malicious robot) can disrupt the coordination strategy of the entire swarm. Therefore, a versatile swarm robotics framework that addresses security issues in inter-robot communication and coordination is urgently needed. In this paper, we show that security issues can be addressed by setting up a token economy between the robots. To create and maintain the token economy, we use blockchain technology, originally developed for the digital currency Bitcoin. The robots are given crypto tokens that allow them to participate in the swarm's security-critical activities. The token economy is regulated via a smart contract that decides how to distribute crypto tokens among the robots depending on their contributions. We design the smart contract so that Byzantine robots soon run out of crypto tokens and can therefore no longer influence the rest of the swarm. In experiments with up to 24 physical robots, we demonstrate that our smart contract approach indeed works: the robots can maintain blockchain networks and a blockchain-based token economy can be used to neutralize the destructive actions of Byzantine robots in a collective-sensing scenario. In experiments with more than 100 simulated robots, we study the scalability and long-term behavior of our approach. The obtained results demonstrate the feasibility and viability of blockchain-based swarm robotics.

One-Sentence Summary

A token economy implemented via blockchain-based smart contracts allows robot swarms to neutralize harmful Byzantine robots.

Introduction

A robot swarm is a multi-robot system consisting of many autonomous robots whose self-organized collective behavior results from local interactions of each robot with its peers and with the environment (*1–6*). The deployment of robot swarms in the real world has been identified as one of the ten grand challenges of robotics for the next decade (*7*). Once they are deployed in the real world, however, robots in a swarm will be subject to events that can harm the swarm's collective behavior: the robots in the swarm could malfunction, could become fully nonoperational, or might be hacked by malicious agents. In addition, peer-to-peer messages could be manipulated while traveling through the network (*8*).

An original and pivotal assumption of the swarm robotics research field has been that the large number of robots and their decentralized organization are sufficient to attain both robustness against malfunctioning robots and security against the presence of malicious robots, thanks to the property of fault tolerance by redundancy (*9, 10*). Indeed, a large body of research has shown that the effects of malfunctioning robots on swarm behavior can be managed (*11–18*). However, this is not the case in the presence of malicious robots: recent research has shown that even a very small minority of such robots can disrupt the functioning of the whole swarm (*19, 20*).

Although security issues are of paramount importance for the deployment of robot swarms in the real world, only a few research papers have been published on this topic so far. One recent important contribution has proposed a method to give commands to robots in a swarm without revealing the overall goal of the mission (*21*). A cooperative robot mission is encapsulated in a Merkle tree, a data structure in which the information in the nodes consists of cryptographic hashes. It thus becomes possible to provide the blueprint of a robot mission without disclosing the raw information describing the mission itself. Information about the swarm's goals cannot be acquired by listening to robot communications or physically capturing a robot. Another recent important contribution has shown that a probabilistic component in the consensus algorithm can minimize the effects of deceitful messages in a binary decision-making scenario (*22, 23*). This damage mitigation occurs implicitly; that is, the approach minimizes the harmful effects without determining whether or how many misbehaving robots are in the swarm. In practice, security research in swarm robotics is in its infancy and further research on this subject is urgently needed, as mentioned in a number of relatively recent articles (*6, 8, 24, 25*).

An important concept for cybersecurity research is the notion of Byzantine agents (*26*) or Byzantine robots (*19*). We call an agent or a robot Byzantine if it shows a discrepancy between its intended behavior and its actual behavior. Such a discrepancy can be a result of programming errors, failed components, or malicious attacks (*27*).

In peer-to-peer networks, the problem of Byzantine agents has been successfully addressed by blockchain technology. This technology combines cryptographic algorithms and decentralized consensus protocols to allow agents to securely share and store data in networks that may contain Byzantine agents, as is the case in robot swarms. The Bitcoin protocol (*28*) was the first successful implementation of a tamper-proof decentralized digital currency. In the Bitcoin protocol, a blockchain hosts a decentralized ledger of Bitcoin transactions. Other blockchain protocols, most notably Ethereum (*29*), have extended the decentralized ledger to a decentralized computing platform. In these protocols, besides financial transactions, programming code can also be stored in the blockchain, and the agents in the network can reach consensus on the outcome of these programs. Programs that are stored on a blockchain and executed using a decentralized computing platform are called blockchain-based smart contracts (or smart contracts, in short form).

Initially, the combination of swarm robotics and blockchain technology was proposed at the conceptual level by describing various potential applications (such as data integrity, consensus agreement, and secure communication) (*30*), but without implementation or empirical testing. More recently, there have been several proofs of concept in which robot swarms and other multi-robot systems interact via smart contracts (*19, 31–37*). The main contributions of these simulation studies were to show that smart contracts can coordinate the robots' behavior and can successfully neutralize Byzantine robots, whereas existing consensus protocols (*38–40*) might break down in the presence of even a single Byzantine robot. However, only small simulated swarms were studied or the blockchain protocol was executed on external infrastructure, and most studies used the widespread proof-of-work protocol (*41*), which requires high computational power and is therefore inappropriate for the relatively simple robots used in most robot swarms. Until now, it has been unclear to what extent blockchain and smart contracts could be used to control large swarms of real robots.

In this article, we go beyond the existing research and present a comprehensive study with real robots (Movie 1). We perform the experiments using Pi-puck robots (*42*), which are e-puck robots augmented by a Raspberry Pi Zero W computer (*43*). In order to allow the robots in the swarm to exchange blockchain information in a decentralized way, we implement a proof-of-authority blockchain (see Supplementary Methods, "Proof-of-authority consensus algorithm") and a mobile ad-hoc network using the Pi-puck robots. In addition, we extend our existing simulation framework (*33*) to handle a large number of simulated robots (see Supplementary Methods, "Simulation framework: Installation and execution of the software"). We first vali-

Movie 1. Overview of the research and of the experiments. The video (`https://youtu.be/9vv0eX5-Q58`) briefly illustrates the working of our blockchain-based robot swarms and the obtained results.

date our simulation by showing that the results in simulation and reality do not differ substantially from each other; we do so by conducting experiments that involve different arena sizes and swarm sizes (up to 24 real and simulated robots). We then use the validated simulator to study the scalability of blockchain-based robot swarms (up to 120 robots) and their behavior over longer experiment times (up to 600 minutes). In order to facilitate the future prototyping and development of a range of blockchain-based applications, we provide both the real-robot framework and the simulation framework as open-source software (see Supplementary Methods, "Simulation framework: Installation and execution of the software" and "Physical Pi-puck robots: Installation and execution of the software").

In our experiments, each robot is a node in a peer-to-peer blockchain network that is maintained by the robots in the swarm (see Materials and Methods). The size of the robot swarm is fixed at the beginning of each experiment (see Supplementary Methods, "Proof-of-authority consensus algorithm", for a brief discussion of swarms that can change their size at runtime). The blockchain protocol that we chose to execute on our robots is Ethereum (*29*), one of the most widely used and mature of the protocols that support smart contracts. For robot swarms, one great advantage of Ethereum is that it allows the usage of the proof-of-authority consensus protocol, instead of the more computationally expensive proof-of-work consensus protocol. Proof-of-authority requires a majority of preselected nodes (that is, a majority of robots in this work) to agree on the state of the blockchain database. We argue that Ethereum's proof-of-authority is an appropriate and energy-efficient choice for robot swarms in which the robots have limited computational power and battery life. Our experimental results show that lim-

ited robots can reliably execute the Ethereum software and the associated proof-of-authority consensus protocol.

Using both real and simulated robots, we study the behavior of robot swarms in the presence of Byzantine robots in a collective-sensing scenario. In this paper, we focus on a specific type of Byzantine fault: the dissemination of wrong or deceitful messages to other members of the robot swarm. This type of fault is harder to detect and more severe than complete robot failures, because it is in general not easy to understand whether a specific robot is or is not trustworthy (*11*), and because deceitful messages can have a strong influence on the overall behavior of the robot swarm (*33*).

Our robots communicate with each other using a smart contract that is executed on the blockchain. The smart contract enables a token economy to be established among the robots in which each robot owns crypto tokens that it can spend to participate in security-critical swarm activities by sending transactions to the smart contract. In our scenario, robots send transactions that contain information extracted from their sensor readings. This information is used by the smart contract to enable the swarm to estimate an environmental property. The smart contract repeatedly provides, at exponentially increasing time intervals, crypto tokens to every robot that is part of the experiment, regardless of how the robot performs. We call this crypto token allocation procedure universal basic income (UBI) (see Materials and Methods for an explanation of the smart contract and of the UBI mechanism). Thanks to the UBI, all robots—at least initially—can send transactions. However, the smart contract also ensures that non-Byzantine robots are rewarded for sending "good" transactions, whereas Byzantine robots do not get rewards for sending "bad" transactions, which results in the Byzantine robots running out of crypto tokens and, therefore, in the impossibility for them to send transactions and influence swarm behavior. In our experiments, a "good" transaction is one that is not rejected by the outlier detection algorithm implemented in the smart contract (see Materials and Methods), whereas a "bad" transaction is a rejected one.

We analyze the performance and resilience of our approach when facing diverse Byzantine faults. We also analyze its scalability and long-term behavior (up to 600 minutes for single runs) using different swarm and arena sizes in simulation experiments.

Results

We evaluate our blockchain-controlled robot swarms both in simulation and with real robots (see Supplementary Methods, "Simulation framework: Installation and execution of the software" and "Physical Pi-puck robots: Installation and execution of the software" for setup and operation instructions). The simulation is programmed to reflect as closely as possible the conditions of the real-robot experiments (see Materials and Methods) and is used to run experiments

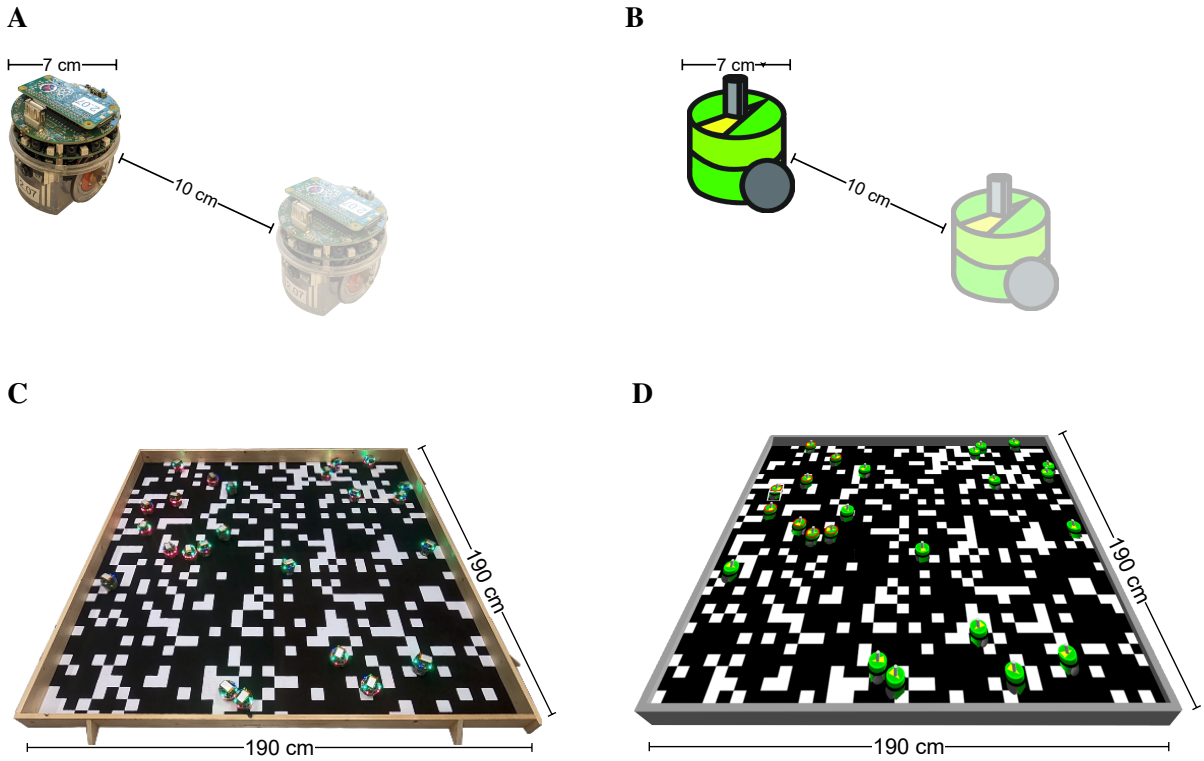

Fig. 1. Overview of the experimental setup. The goal of the robot swarms is to estimate the fraction of white floor tiles in a checkerboard environment (see also Movie 1 for an overview of the experiments). Reaching the goal is complicated by the presence of Byzantine robots that send incorrect sensor data. The robots use blockchain-based smart contracts to coordinate with each other. Each robot in the swarm is a node in the decentralized blockchain network and relies only on local communication (communication range: 10 cm). The experiments are conducted both in reality, using Pi-puck robots (*42*); and in simulation, using the ARGoS robot swarm simulator (*44*). (**A**) The Pi-puck robot and (**B**) its simulation model. (**C**) The real checkerboard environment and (**D**) the corresponding simulated environment.

in situations that are challenging to study with real robots due to time, budget, battery, and space limitations.

In all experiments, the goal of the swarm is to reach a consensus on the percentage of white tiles (a value that can be set between 0 % and 100 % by the experimenter), in an environment in which the floor is covered with white and black tiles (Fig. 1). To obtain their individual estimates, robots equipped with ground sensors perform a random walk in the environment.

To reach consensus on a value that is as close as possible to the correct one (25 % white tiles in all our experiments), the robots' estimates are broadcast (communication range: 10 cm) by the robots in the form of transactions. The decentralized blockchain protocol ensures that the

Table 1. Overview of the experiments and their parameters. When an experimental setting is underscored, it means that the corresponding experiment was performed both with real robots and in simulation; when it is not underscored, it means that the experiment was performed exclusively in simulation.

Experiment	Swarm size	Number of Byzantine robots	Arena size [m^2]
1 – Robustness against an increasing number of Byzantines			
a – Byzantine fault: 0 % white tiles	$\underline{24}$	$\underline{0}, \underline{3}, \underline{6}, \underline{9}$	$\underline{3.6}$
b – Generality study	24	$0, 3, 6, 9$	3.6
2 – Scalability and long-term behavior			
a – Increasing swarm size in constant arena	$\underline{8}, \underline{16}, \underline{24}$	$\underline{0\,\%}, \underline{25\,\%}$	$\underline{3.6}$
b – Increasing arena size with constant robot density	$\underline{8}, \underline{16}, \underline{24}; 48, 72, 96, 120$	$\underline{25\,\%}$	$\underline{1.2}, \underline{2.4}, \underline{3.6}; 7.3, 10.9, 14.4, 18.1$
c – Long runtime: swarm estimate behavior and blockchain growth	$8, 16, 24; 48, 72, 96, 120$	$25\,\%$	$1.2, 2.4, 3.6; 7.3, 10.9, 14.4, 18.1$
d – Long runtime: token flow dynamics	24	$25\,\%$	3.6

robots are able to agree on the addition of transactions to the blockchain. The smart contract then aggregates them, produces the collective estimate, and determines when the collective estimate has converged (see Materials and Methods).

The swarm estimation activity is complicated by the fact that a part of the swarm consists of Byzantine robots that send wrong estimates. Depending on the experiment, the Byzantine robots either constantly send '0 % white tiles' or they draw their individual estimates from a random distribution (either a Bernoulli distribution with $p = 0.5$ or a uniform distribution).

To study the performance of our system we run two experiments (see also Table 1). The first experiment investigates whether the presented system tolerates an increasing number of Byzantine robots in the swarm. In this experiment we test our system under the conditions of different Byzantine faults and floor layouts. The second experiment addresses the scalability of the approach and its long-term behavior. Regarding scalability, we study both how the swarm performance changes when the swarm size increases in an arena of constant size and therefore the robot density increases, and when the size of the arena increases proportionally to the swarm size such that the robot density remains constant. We then perform longer-term experiments in which we let our system run for up to 600 minutes (a 40-fold increase with respect to the previous experiments) and study the behavior of the swarm estimates generated by the robots, the growth of the blockchain size, and how the crypto tokens become distributed among robots.

The performance of the presented system is evaluated by calculating the absolute error of the swarm, that is, the absolute difference between the actual percentage of white tiles and the swarm estimate at the end of each run. The swarm estimate is also compared to a baseline that is calculated offline by aggregating the individual sensor values without outlier detection and without using a blockchain (see Materials and Methods). Furthermore, depending on the exper-

iment, we assess performance according to the following metrics (see Materials and Methods for definitions): convergence time (the time it takes until the smart contract determines that the robots have agreed on an estimate), the size of the blockchain in megabytes, the quantity of crypto tokens owned by each robot, and the bandwidth utilization.

Experiment 1: Robustness against an increasing number of Byzantines

In this experiment, we study the influence Byzantine robots have on the performance of a swarm of fixed size $N = 24$ robots. The number of Byzantine robots in the swarm is increased from 0 to 9 in steps of 3, that is, they represent from $0\,\%$ to $37.5\,\%$ of the whole swarm in steps of $12.5\,\%$. We consider a few different experimental conditions.

First, in Experiment 1a, we consider the situation in which Byzantine robots always send '$0\,\%$ white tiles,' independent of their actual sensor readings. Our hypothesis is that the lowest absolute error is obtained when there are no Byzantine robots in the swarm.

The results (see Fig. 2A) show that when there are no Byzantine robots, the median absolute error is very low for all approaches (reality: $0.6\,\%$, simulation: $0.6\,\%$, baseline: $0.5\,\%$). The low error, together with the low variance of the error, shows that our setup is functional: the ground sensors work as intended, the random walk routine is appropriate, and, most importantly, our blockchain approach performs as well as the baseline. This means that even when there are no Byzantine robots in the swarm, a blockchain approach can be appropriate because it does not degrade performance. This is important because in general we do not know a priori whether the swarm will contain Byzantine robots.

When the number of Byzantine robots increases, the median error of our approach increases only slightly. By contrast, the baseline shows an approximately linear increase in the error (Fig. 2A). The difference occurs because the estimates of the Byzantine robots are rejected by the smart contract and therefore the Byzantine robots eventually run out of crypto tokens and can no longer influence the collective estimate.

The presence of Byzantine robots also slows down the convergence process (Fig. 2B). Indeed, when the number of Byzantine robots increases, the number of valid estimates—those that are not rejected by the smart contract—in a given time period decreases, because the smart contract filters out the estimates sent by the Byzantine robots. Therefore, a longer period of time is necessary for the swarm estimate to converge.

Second, in Experiment 1b, we investigate the generality of our approach by checking if the results obtained in the previous experiment hold when varying some of the experimental conditions (these tests are run in simulation only). We consider the following three conditions: First, for each repetition of the experiment, a new layout for the tiles on the floor is randomly generated (see Materials and Methods) and the Byzantine robots always send '$0\,\%$ white tiles.' Second, every time a Byzantine robot sends an estimate, it selects the estimate to be either '$0\,\%$

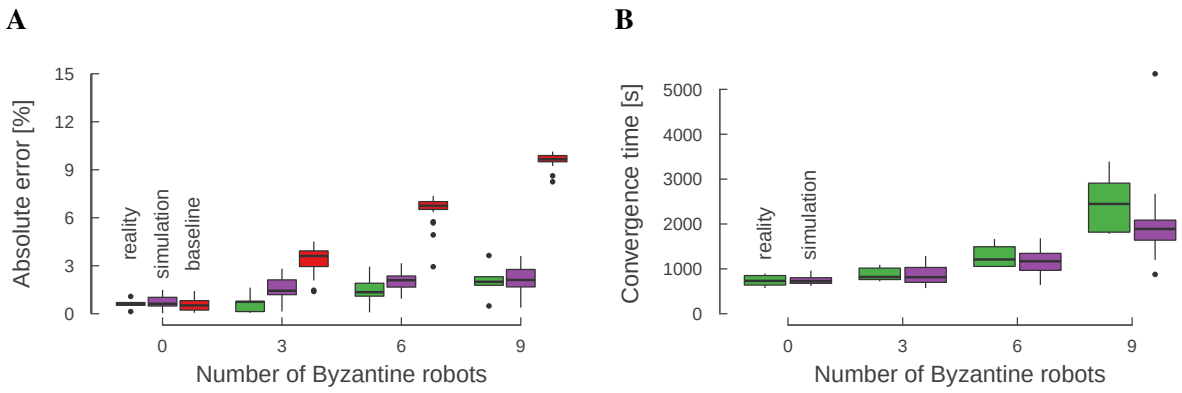

Fig. 2. Experiment 1a – Increasing Byzantines: 24 robots in total, of which 0, 3, 6, or 9 are Byzantine robots that always send a 0 % estimate. In all the plots presented in this paper, the boxes in the boxplots represent the first quartile, median, and third quartile; the whiskers extend up to the 1.5 interquartile range values; and the dots represent the outliers. The green boxes are the results for real robots, the purple boxes are the results for the simulations, and the red boxes are the results for the baseline (see Materials and Methods for the calculation of the baseline). For each setting of each experiment, we conduct 5 repetitions with real robots and 20 repetitions in simulation. (**A**) The approach works well both in reality and in simulation and the median of the absolute error stays below 2.2 % even with 9 Byzantine out of 24 robots (that is, 37.5 % of the robots in the swarm are Byzantines). (**B**) The higher the number of Byzantines, the stronger is their influence on the convergence time. Convergence time is not reported for the baseline as it is computed offline.

white tiles' or '100 % white tiles' with the same probability (the estimate is selected using a Bernoulli distribution with $p = 0.5$). Third, every time a Byzantine robot sends an estimate, it selects the estimate to be 'x % white tiles', where x is a random value drawn from the uniform distribution between 0 and 100.

As can be seen in Fig. 3, the results are very similar to those of the previous Experiment 1a, which suggests that our approach can handle a range of different tile distributions and Byzantine faults.

Experiment 2: Scalability and long-term behavior

In the second experiment, we first study how our approach scales when the swarm size is increased while the arena size remains constant (Experiment 2a), as well as when the size of the arena grows proportionally to the swarm size (Experiment 2b). We then consider much longer runtimes to study how the swarm's collective estimate behaves after convergence is reached—that is, whether the collective estimate gets more accurate, the estimate stagnates, or even,

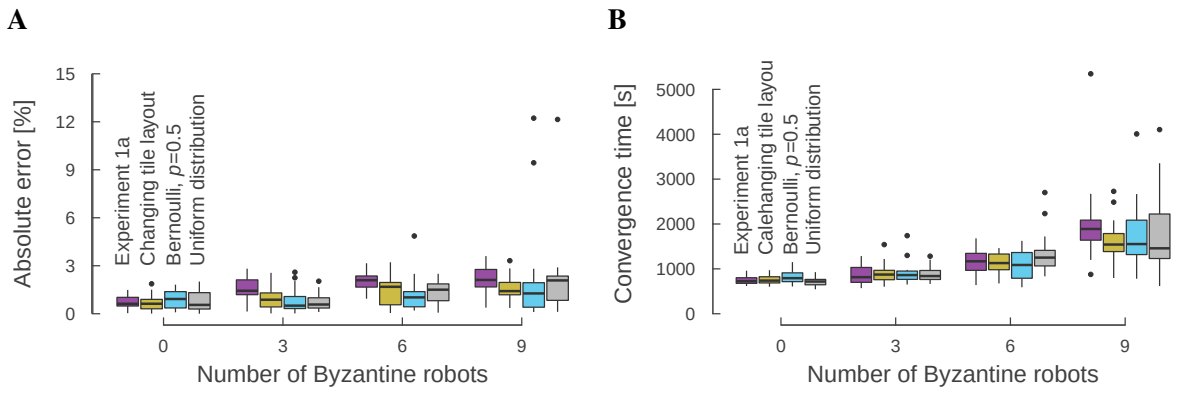

Fig. 3. Experiment 1b – Generality study: 24 simulated robots in total, of which 0, 3, 6, or 9 are Byzantine robots. For each setting of each experiment, we conduct 20 repetitions in simulation. To better compare the results, we show again those from the previous Experiment 1a (purple boxes), in which robots always send a 0 % estimate. The ocher boxes show the results for different floor layouts, in which robots always send, as in Experiment 1a, a 0 % estimate. The light blue and gray boxes respectively show the results for the Byzantine faults where the estimates are drawn from the Bernoulli distribution with $p = 0.5$ and for the Byzantine faults where the estimates are drawn from the uniform distribution; in both cases the floor layout is the same as in Experiment 1a. (**A**) Absolute error. (**B**) Convergence time.

whether convergence is lost (Experiment 2c)—and to study the dynamics of the token economy and the long-term consequences of sending wrong data (Experiment 2d). In all experiments, the Byzantine robots send an estimate of $0\,\%$ white tiles.

In Experiment 2a, we investigate to what extent the size of the swarm has an influence on the absolute error and the convergence time. In particular, we are interested in understanding whether a larger swarm yields more accurate results and whether this may be at the expense of a longer convergence time. In addition, we are interested in studying how the approach performs when connectivity is sparse. To this end, we use three swarm sizes: 8, 16, and 24 robots in an arena with a constant size of $3.61\,\mathrm{m}^2$. We perform the experiments both with and without Byzantine robots ($25\,\%$ when present).

Results show (see Fig. 4A and Fig. 4C) that, if there are no Byzantine robots in the swarm, the median of the absolute error is low: under 1.0 % for all swarm sizes and approaches. When using the blockchain-based approach, the median of the absolute error remains low (less than 2.1 %) even in the presence of Byzantine robots, whereas it grows to more than 6.5 % in the baseline.

The results additionally show that the approach still converges when the swarm is only sparsely connected. In particular, in the experiment with 8 robots in the $3.61\,\mathrm{m}^2$ area, connectivity is very sparse. Therefore, most of the time, the robots do not have any connections to other

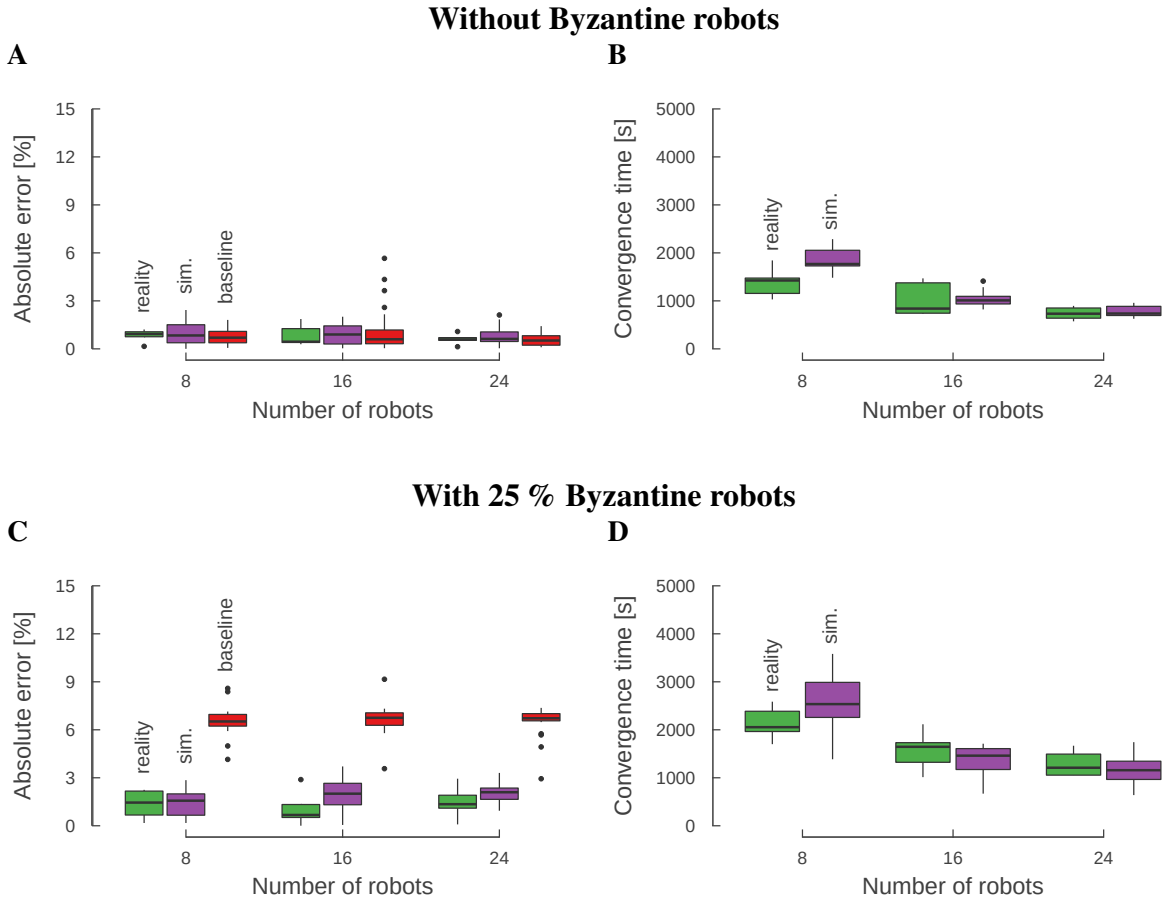

Fig. 4. Experiment 2a – Increasing swarm size in an arena of constant size: 8, 16, and 24 robots in total without and with 25 % Byzantine robots that always send a 0 % estimate. For each setting of each experiment, we conduct 5 repetitions with real robots and 20 repetitions in simulation. (**A** and **C**) Regardless of swarm size, the absolute error of our blockchain-based approach is small, even in presence of Byzantine robots. (**B** and **D**) The convergence time decreases noticeably with a larger swarm size. This is due in part to the better connectivity and in part to the higher accuracy when more robots make a measurement. The decrease of the convergence time as a result of a larger swarm is an indication of the scalability of our approach. Convergence time is not reported for the baseline as it is computed offline.

robots. The sparse connectivity leads to many network partitions that are eventually resolved.

The convergence time tends to decrease as the swarm size increases (see Fig. 4B and Fig. 4D). The main reason for the shorter convergence time is that the average connectivity—that is, the average number of other robots to which a robot is connected in the time unit—is higher in larger swarms: it is 0.24 robots per second in swarms with 8 robots, 0.59 robots per

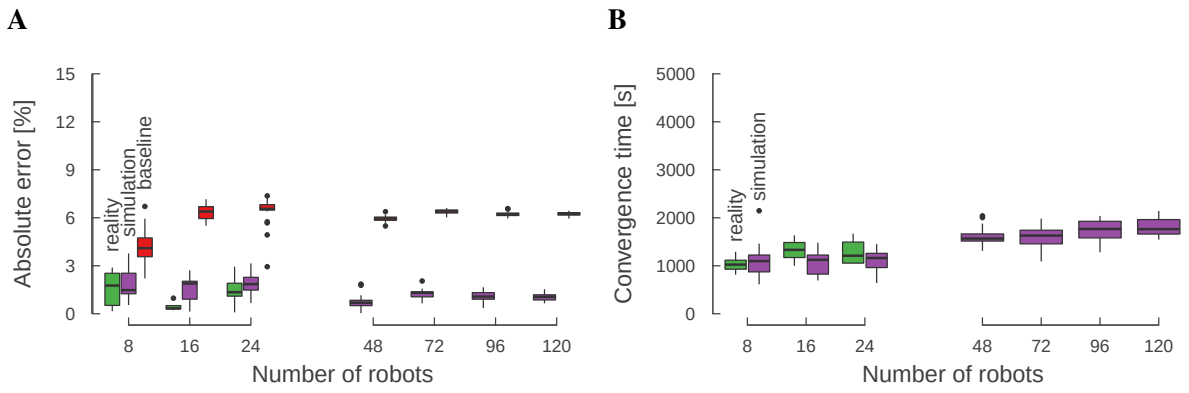

Fig. 5. Experiment 2b – Increasing arena size with constant robot density: up to 120 simulated robots of which 25 % are Byzantine robots that always send a 0 % estimate. The x-axes are interrupted to emphasize the fact that, for swarm sizes larger than 24, the number of robots in the swarm is increased in steps of 24 instead of 8. For the experiments with real robots, we run 5 repetitions for each setting; for the simulations, we run 20 repetitions. Experiments with 48 or more robots are run in simulation only. (**A**) Independent of the swarm size, the absolute error is small. With an increasing swarm size, however, the variability decreases, so that the confidence in the swarm estimate is increased. This result supports the scalability of the approach. (**B**) The convergence time increases only slightly with an increasing swarm size: from 8 robots to 120 robots, the convergence time less than doubles.

second with 16 robots, and 0.91 robots per second with 24 robots. In addition, a larger swarm has more sensor readings, resulting in a lower variance of the sample mean and thus a shorter convergence time.

In Experiment 2a, we showed that a larger swarm yields both a lower absolute error and a shorter convergence time when the arena size is kept constant. In Experiment 2b, we study whether the same holds true when the arena size increases proportionally with the swarm size and therefore the robot density, measured as number of robots per m^2, remains constant (see Materials and Methods for exact arena sizes). Results show (see Fig. 5A) that in this case, in all experiments, the median of the absolute error stays below 3 %. The larger the number of robots, the more accurate the estimate: both variance and absolute error slightly decrease with an increasing number of robots. Once again, this is a good indication of the scalability of the approach.

The convergence time (see Fig. 5B) increases with the swarm size, but this increase is small: from an 8-robot swarm to a 120-robot swarm (a factor of 15), the median of the convergence time less than doubles. The increase in convergence time happens because in a very large arena, despite a constant robot density, longer disconnection times can occur (for example,

when almost all robots are in one corner and a single robot is in another corner), and it can therefore take longer for all robots to receive the convergence signal.

Overall, the low absolute error for all swarm sizes and the small increase in the convergence time further demonstrate the scalability of the approach and suggest that a blockchain-based approach might also work in real-world deployments with much larger arenas and higher numbers of robots.

Experiments 2c and 2d are designed to test the long-term behavior of our system. Therefore, we do not stop the experiments after the swarm has converged to a common estimate; instead, we let them continue to run for a total duration of 600 minutes. We run these experiments in simulation only, due to the limited battery life of currently available robots.

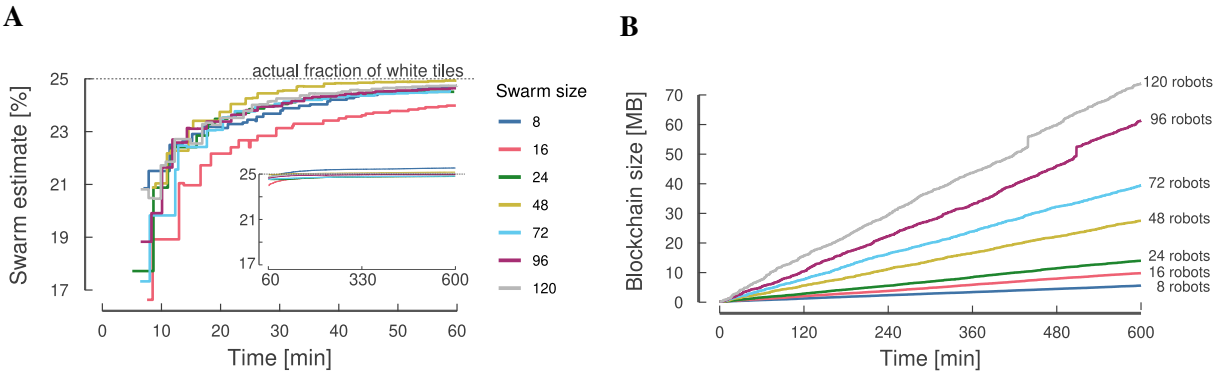

Fig. 6. Experiment 2c – Long runtime: swarm estimate behavior and blockchain growth: up to 120 simulated robots of which 25 % are Byzantine robots that always send a 0 % estimate. (**A**) The swarm converges fast and the estimate is accurate, as already shown in the previous experiments. The inset shows the swarm estimate from 60 min to 600 min. After 60 minutes, the swarm estimate hardly changes. (**B**) With an increasing swarm size, the blockchain size also increases faster. This is because the robots create more transactions in a larger swarm. The sudden increases in blockchain size for 96 and 120 robots (approximately at time 500 min and 440 min, respectively) are due to the compaction algorithm of Ethereum's database implementation (LevelDB) and are offset by a later decrease in blockchain size when the blockchain reaches approximately 120 MB. The presented data are obtained with one run in simulation for each swarm size.

Fig. 6 shows the results of Experiment 2c, in which we consider swarms of sizes 8 to 120 robots. 25 % of the robots are Byzantine and the runtime is 600 minutes. We analyze individual runs in this experiment: the reported data represents one repetition for each swarm size. At the beginning of the experiments, the robots exclusively rely on the UBI in order to pay for sending their estimates. Therefore, there is a delay before the first swarm estimate is generated (Fig. 6A)

as the robots can send their individual estimates only after they have received enough UBI payments. Note that the time it takes to generate the first swarm estimate is largely independent of the swarm size N, because, as we scale the arena size proportionally to N (ensuring a constant robot density), the block time (see Supplementary Methods, "Proof-of-authority consensus algorithm") and the initial delay in the transfer of the UBI payments do not change substantially. After the first swarm estimate is generated, the estimate quickly converges to the actual fraction of white tiles (Fig. 6A). The estimate converges from lower values, because the Byzantine robots, which always send a $0\,\%$ estimate, initially shift the estimate down before gradually losing influence.

Results also show (see Fig. 6B) that the blockchain size grows linearly with time, which makes it possible to make good predictions about the evolution of the blockchain size over longer runtimes. The rate of growth of the blockchain also increases with the swarm size, because a larger swarm creates more transactions.

Finally, in Experiment 2d we analyze the inbound token flow over time—that is, the cumulative sum of crypto tokens received by a robot as a function of time, which consists of UBI payments and reward payments. Fig. 7A demonstrates the dynamics of the blockchain token economy and the sound operation of the smart contract logic: the longer the experiment lasts, the greater the difference between the fraction of inbound token flow attributed to Byzantine and non-Byzantine robots. This is because over time the UBI payments, which are received by all robots, become less and less frequent and robots obtain crypto tokens mainly from the reward payments (see Materials and Methods), which are mostly earned by non-Byzantine robots (this is because non-Byzantine robots' estimates are often accepted and rewarded by the smart contract, whereas those of Byzantine robots tend to be discarded). Note that, because of this mechanism, all the UBI payments, which are distributed to all robots at exponentially increasing time intervals, are eventually redistributed to the non-Byzantine robots. Therefore, the non-Byzantine robots' inbound token flow increases logarithmically over time, and so does the rate with which they submit their estimates (see Fig. S1 and Supplementary Results, "Rate of estimates").

Fig. 7B shows how the inbound token flow changes over time for each robot in the swarm. In particular, it can be observed that the inbound token flow of the Byzantine robots stays low and only increases when they receive UBI payments. It can also be observed that token flow can vary greatly among non-Byzantine robots (two of the non-Byzantine robots have a substantially lower token flow, marked with an asterisk in Fig. 7B). The lower token flow is a result of random fluctuations: at the beginning of the experiment, the robots' estimates are still inaccurate as they have only taken a few ground sensor readings. It can therefore happen that they send inaccurate estimates early in the experiment (as done by the two robots marked with the asterisk in Fig. 7B). In this case, the smart contract discards their estimates and does not

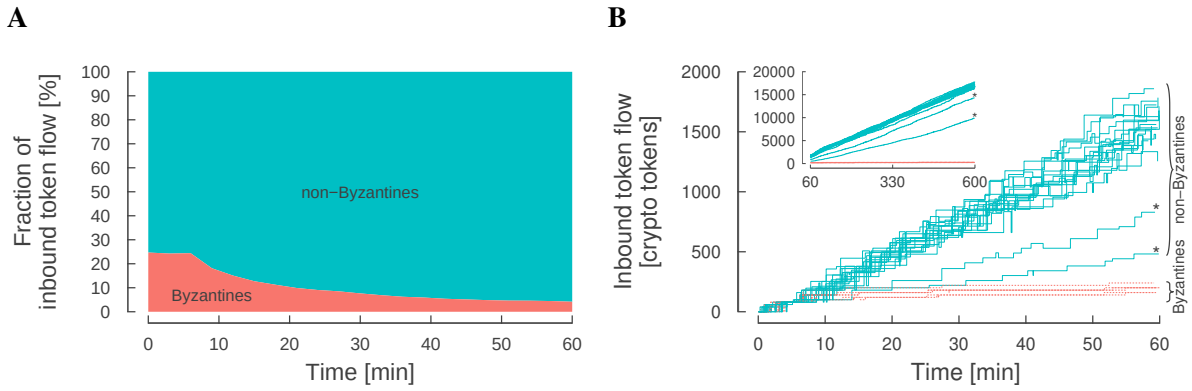

Fig. 7. Experiment 2d – Long runtime: token flow dynamics: 24 robots of which 25 % are Byzantine robots that always send a 0 % estimate. (**A**) The plot shows the fractions of the inbound token flow received over time by the non-Byzantine robots (cyan area) and Byzantine robots (red area). In the beginning of the experiment, all robots receive the UBI payments. As the Byzantine robots constitute 25 % of the swarm size, they receive 25 % of the inbound token flow. However, later on, reward payments are the major source of token flow; due to the outlier detection mechanism, Byzantine robots do not receive reward payments and their fraction of the token flow decreases. After 60 minutes the token flow attributed to Byzantine robots continues to converge towards 0 %. The plot is generated by running 20 repetitions in simulation; standard deviation is not visible because always smaller than 1 % of the fraction of inbound token flow. (**B**) The plot shows the inbound token flow per robot over time (each curve represents one robot), for one run with 24 robots. The inset shows the token flow from 60 min to 600 min. Whereas the non-Byzantines (cyan curves) continuously receive payments, the Byzantines (red curves) receive payments only when they receive the UBI, which happens rarely and therefore their token flow remains very low. Note that there are also two non-Byzantine robots that have a substantially lower token flow (marked with an asterisk in the figure) than other non-Byzantine robots as they sent inaccurate estimates at the beginning of the experiment.

reward them, even though they were not Byzantine robots. These initial inaccuracies can then lead to an overall lower token flow for the concerned robots.

Discussion

The benefits of using blockchain technology in robot swarms

Thanks to blockchain technology's solution to the double-spending problem (the problem of the same token being spent multiple times simultaneously) which allows decentralized systems to maintain scarce digital quantities in the form of crypto tokens, the robots in our swarm are able

to perform economic transactions.

In a series of experiments, we show that our blockchain-based token economy can be used to neutralize Byzantine robots. Indeed, when Byzantine robots are part of the swarm, our blockchain-based approach always performs better than the baseline, and when there are no Byzantine robots in the swarm, the blockchain-based approach still performs as well as the baseline. As we cannot know a priori whether Byzantine robots will or will not be present in a swarm, this assures that the obtained performance will be very good in both the presence and absence of Byzantine robots.

The smart contract calculates the swarm estimate and determines convergence online—that is, during the experiment and within the swarm, letting the swarm determine in a fully decentralized way if the shared estimate has converged. Importantly, this can be done even in the presence of Byzantine robots. The swarm can therefore autonomously decide when to switch to the next task without the need for any external interactions.

In addition, there are two desirable properties that we inherit from the fact that we are using blockchain technology. First, during the autonomous operation of the robot swarm, a conflict-free logbook of the messages exchanged is automatically stored in the blockchain. This logbook can be used for establishing accountability and for performing postliminary investigation and data forensics (*25*), as the blockchain offers non-repudiation (a robot in the swarm cannot deny that it has sent a message). It is important to note that, when using our blockchain-based approach, in order to obtain data about the entire course of the experiment, it is sufficient to extract data from only one robot, as all robots in the swarm—Byzantines included—share the same logbook (this is particularly useful when a majority of the robots are lost or destroyed).

Second, when using blockchain technology, implementing a collective behavior amounts to programming and deploying a smart contract, rather than re-programming every single robot in the swarm. Because the control information is encapsulated in one place, it becomes easier for a user both to modify the swarm behavior and to exchange confidential information with the swarm using approaches as the one presented in (*21*).

Computational requirements

The addition of a blockchain protocol layer to the control of a robot swarm increases the complexity of the system and the computational load on the robots. An important question is therefore whether a blockchain is necessary to secure a robot swarm, or if a subset of blockchain technology's components would be sufficient. According to the current state of the art, the answer is that the components of blockchain technology are the minimum possible set that is capable of creating a tamper-proof system in a network of mutually untrusting agents (*45*). These components are: public-key cryptography and the use of digital signatures, a consensus proto-

col, a decentralized database, and, if the system should be used as a computing platform, smart contracts. Omitting any of these components would lead to a system that lacks the required security for real-world deployment, in the following respective ways: lack of message authentication and non-repudiation, inability to solve conflicting states of the decentralized ledger, lack of a tamper-proof register of events accessible during and after an experiment, and inability to guarantee agreements on the outputs of programs.

Given that we need each of these components, the important remaining question is whether implementing a blockchain protocol layer is feasible in terms of computational load. In other words, we were interested in understanding whether, when executing blockchain software on real robot swarms, the computation, storage, and communication requirements of the blockchain protocol are manageable by a system composed of robots that have limited computational power, low storage capacity, and rely on local and dynamic communication. Our experiments have demonstrated that this is indeed possible.

The best-known and most common consensus protocol used in blockchain technology is proof-of-work, first proposed for Bitcoin. An important drawback of this protocol is that it might require a huge amount of energy and can therefore be problematic for use in a robot swarm (*41*). Our choice was therefore to employ the proof-of-authority consensus protocol, which uses much less energy and that we found to be well-suited for the simple robots used in our experiments. With proof-of-authority, the CPU and RAM capacities of the robots were far from fully utilized (see Supplementary Results, "CPU and RAM utilization").

Concerning storage of the blockchain in the robots, we have shown (Fig. 6B) that the blockchain size grows linearly both with the runtime and with the swarm size. Using a swarm of 120 robots, the size of the blockchain data folder in a 600-minute run was 73.6 MB. As the Pi-puck robots we use are equipped with a 16 GB SD card, the same experiment could be executed for several weeks before the robots run out of storage. For this reason, we do not consider storage requirements to be a consequential issue for our system.

In order to get an estimate of the blockchain's communication overhead, we measured the bandwidth utilization (see Materials and Methods for the definition of bandwidth utilization) for swarms of 120 robots. The mean bandwidth utilization was 2.11 KB/s during the entire experiment. If one considers only the time when the robots communicated with each other, the mean bandwidth utilization was 4.97 KB/s. During the 600-minute experiment, the total size of sent packets per robot was 72.5 MB (the size of sent packages is smaller than the final blockchain size, which is 73.6 MB, as the blockchain data folder also includes files needed for the database workings). The results indicate that communication costs are rather low: the total size of sent packets is approximately the same size as the final blockchain size and the bandwidth utilization does not increase over time. As a consequence, the bandwidth of the Pi-puck's onboard Wi-Fi module is far from being fully utilized (less than 1 % in our experiments).

Challenges

More complex application scenarios and smart contracts. A future challenge will be to use the technology presented in this paper to address situations in which robot swarms need to perform more complex tasks or are faced by smarter Byzantine robots. In these cases, there will be a need for more sophisticated smart contracts. In this paper, our goal was to build the foundations for the use of smart contracts and blockchain technology in robot swarms: first, by bringing blockchain technology to real robots using the proof-of-authority consensus protocol and a suitable communication architecture, and second, by studying Byzantine fault-tolerance, economic rewards, scalability, and sparse connections. To do so we chose to use a simple application scenario and associated smart contract.

For example, the task given to the swarm—that is, the estimation of the tile fraction—is relatively simple, and could be performed by a single robot. However, even in this simple case, our blockchain-based robot swarm is desirable as it is robust to the presence of Byzantine robots. By contrast, in the single robot case, if the robot were Byzantine, we would not have any way to neutralize it.

Also the smart contract that we have designed is rather simple and could be circumvented by smarter Byzantine robots (see Supplementary Discussion, "More sophisticated Byzantine behavior", for examples). Even though it is in principle possible to write smart contracts that take into account any foreseeable behavior of Byzantine robots (this can be done because smart contracts are Turing-complete), we have chosen to implement a relatively unsophisticated smart contract that is however good enough for our proof-of-concept study (see Materials and Methods for details about the smart contract that we developed for this research). In practice, the smart contract design will need to be adapted to different circumstances, including tasks, environments, and possible attacks. For example, if the considered environment is dynamic—as opposed to the static environment studied in this paper—a more sophisticated smart contract could be used that removes older estimates or detects changes in the robots' estimates.

Dynamic swarms. For real-world deployment it might be desirable to have robot swarms where robots can dynamically join or leave or might belong to different owners. However, in such swarms it is easier to introduce Byzantine robots as there is no control over who introduces a robot. Also, it becomes in principle possible to perform a Sybil attack (an attack in which a robot controls several identities) because the total number of robots is not known, as it was the case in the "closed" swarm considered in this paper. One possible approach to this challenge would be to use a permissioned blockchain with proof-of-authority. An issue with such a permissioned network is that an entity in charge of granting permissions needs to be defined. This entity could be the swarm itself, for example using a voting mechanism (*46*) or a vouching system (see Supplementary Methods, "Proof-of-authority consensus algorithm"), or an external entity, such as the owners of the robots belonging to the swarm.

Network partitions. In a real-world deployment, robot swarms could face more severe network partitions (clusters of robots in disconnected sub-swarms) than in our experiments. Such network partitions could delay information propagation and could be exploited by an attacker to gain control of a relative majority of robots in the swarm. There are however ways in which this problem can be addressed.

One possible approach to prevent network partitions would be to let the robots use a self-organized aggregation strategy (as opposed to the random walk behavior used in the present research). One could even develop algorithms that combine blockchain-based consensus protocols and (physical) aggregation of robots: for example, a "proof-of-dissemination" algorithm could reward robots that had the most physical encounters with other robots and, therefore, helped to disseminate information in the swarm. Another approach could be to use messenger robots that have a higher communication range or speed than the rest of the swarm: for example, one could employ drones as messenger robots, whereas the rest of the swarm could be composed of ground robots.

Lastly, one could employ partition-tolerant distributed ledger technology frameworks as they become available. Indeed, it is certainly possible that, as blockchain and distributed ledger technology research is under very active development, in the future, we will see ready-to-use protocols that are more suited for our purposes than Ethereum with the proof-of-authority consensus protocol. For example, partition-tolerant frameworks based on directed acyclic graphs (DAGs) are currently under development and do not require all robots to have an identical copy of a blockchain, and hence smart contracts do not need to be executed by all robots (*36, 47*). As in stationary blockchain networks, there will be no universal solution, but the respective blockchain framework and its parameters must be adapted to the respective application (*48*).

51 % attacks. Even though situations in which a majority of the robots are Byzantine are likely to be rare in real-world applications, it might be desirable to secure robot swarms even in these cases. The security and performance of our approach is however limited to what blockchain technology can provide. For example, proof-of-authority, as any other known consensus protocol, is susceptible to attacks when a majority of participants are Byzantine. In other words, if an attacker can control more than 50 % of the robots in the swarm, the information that is stored in the blockchain is no longer trustworthy. This is a limitation that is inherent to the chosen technology and which might be removed or made less stringent through future research developments in the field of distributed consensus protocols. However, as of today, there are no other distributed consensus protocols that achieve better results.

Time delays. The use of blockchain technology can introduce time delays. After a blockchain transaction is created, it must be inserted into a block, which takes a certain amount of time depending on the connectivity and the selected blockchain parameters. In our particular case, this is regulated by the proof-of-authority consensus protocol. We have chosen the default value

(15 seconds) for the minimum block time. Setting the block time is a trade-off: a lower block time prevents delays if the blockchain is to be used for real-time coordination (*35*). However, it also increases the probability of blockchain forks, which can lead to (temporary) conflicting information and an increase in communication overhead. Setting the block time to higher values prevents forks due to network partitions at the expense of slower consensus agreement. In our research, the presence of delays did not disrupt the swarm behavior because the smart contract did not handle the robots' low-level control. However, in applications where this is not the case (such as in a task allocation scenario where the smart contract tells the robots which tasks to perform), delays need to be considered when deciding which data should be processed by the smart contract. For example, one might choose to let the smart contract process safety-critical data, whereas the processing of time-critical data might be done locally by the robots. As mentioned above in the network partitions challenge, another approach would be to use a different distributed ledger technology framework with higher transaction throughput or better management of network partitions, such as a DAG-based architecture (*49*).

Materials and Methods

Design of the smart contract

Variables and functions

There are four functions in the smart contract with which the robots can interact:

`registerRobot()`: this function is executed by each robot at the start of the experiment; it is required before a robot can use the other functions of the smart contract. When executed, the function increases the swarm size N in the smart contract and registers the robot's public address on the blockchain.

`askForUBI()`: this function allows robots to obtain the UBI by sending a transaction to the smart contract.

`sendEstimate(<localEstimate>)`: this function lets the robots store their local estimates on the blockchain. In order to store an estimate, robots have to send 40 crypto tokens through a transaction.

`hasConverged()`: this function determines whether the absolute difference between the previous and the current value of the swarm estimate is smaller than $0.2\,\%$, in which case it

returns 'true.'

Crypto-token economy

In our experiments, the rules for the distribution of crypto tokens form the basis of an economic mechanism that we designed to minimize the influence of Byzantine robots on the collective estimate. The mechanism limits the number of messages (transactions) a robot can send by attributing an economic cost (in crypto tokens) to them. This works by requiring all the robots to pay a certain number of crypto tokens to send transactions (40 crypto tokens in our experiments), but only non-Byzantine robots are rewarded for this activity and can therefore continue to participate in the token economy, whereas Byzantine robots soon run out of crypto tokens and can no longer send transactions.

The robots can obtain crypto tokens in two ways: by being rewarded when sending accepted estimates and by receiving the UBI. These two ways are described in more detail in what follows.

Rewards and detection of Byzantine robots Every time a robot sends an estimate (using the `sendEstimate(<localEstimate>)` function), this estimate is stored in a temporary list in the smart contract. As soon as the temporary list contains N estimates for the first time (called the first proposal round), the smart contract computes their arithmetic mean which becomes the swarm estimate. The temporary list of estimates is then deleted. From then on, each time this list contains N estimates, the smart contract executes a new proposal round and discards all the estimates that differ more than $\delta = 0.2$ from the current swarm estimate; it then uses the remaining ones to generate the new swarm estimate (obtained as before by computing the arithmetic mean). The temporary list is deleted again afterward. All robots that sent an accepted estimate receive a reward payment that consists of a refund of the 40 crypto tokens required for the `sendEstimate` transaction plus a bonus consisting of an equally distributed share of the non-repaid crypto tokens of the discarded estimates.

Example: All $N = 16$ robots in a swarm have sent an estimate and, thus, there are $16 \times 40 = 640$ staked crypto tokens. Of the 16 estimates, say 13 estimates are accepted, and 3 estimates are discarded because their values differ more than 0.2 from the current swarm estimate. The 13 accepted estimates are used to update the current mean in the smart contract. The 3 robots that sent the non-accepted estimates lose their 40 crypto tokens stake. The 13 robots that sent accepted estimates are refunded their 40 crypto tokens, plus a bonus of about 9.2 crypto tokens ($3 \times 40 = 120$ tokens of the non-accepted estimates distributed among 13 robots).

Universal Basic Income (UBI) The UBI is an economic mechanism that we established within the smart contract to grant crypto tokens to the robots. The UBI functions as follows: the smart

contract is provided with a quasi-infinite number of crypto tokens in the genesis block (these crypto tokens are securely locked and inaccessible). At blockchain block numbers which are a power of 2 (that is, blocks $1, 2, 4, 8, \ldots$), the smart contract transfers 20 crypto tokens to each robot in the swarm when the robots execute `askForUBI()`. This exponential scheme ensures that at the beginning of an experiment each robot receives sufficient crypto tokens to be able to send estimates to the smart contract; however, over time, sending useful information (more precisely, accepted estimates) becomes the main means to receive crypto tokens and to be able to continue participating in the experiment.

The robots

For the experiments with real robots, we use the Pi-puck robot platform (*42*), with a maximum swarm size of $N = 24$ robots. The Pi-puck is an e-puck robot (*43*) extended by a Raspberry Pi Zero W single-board computer. The Raspberry Pi Zero W is a low-cost extension that is able to run a Linux-based distribution. It has a Wi-Fi module and a 1 GHz processor with 512 MB of RAM. For data storage, we use a 16 GB SD card. The Raspberry Pi extension improves the robots' communication capabilities and computational power and, therefore, allows for the implementation of more complex algorithms compared to previous e-puck robot versions. The low cost and size of the Raspberry Pi Zero W combined with its ability to execute the Ethereum software supports our claim that blockchain technology is suitable for swarm robotics applications.

This paper adheres to the traditional assumptions of swarm robotics: resource-constrained hardware, local communication, no access to external infrastructure (such as the Internet and cloud computing), and changing network topologies. As a result, the presented approach can potentially be used in the typically challenging environments where robot swarms are envisioned to be deployed, such as underwater or in space. Parts of this research, in particular economic rewards or Byzantine robot detection, can also be transferred to other multi-robot systems, including systems with access to the Internet or a cloud server.

The communication architecture

For our robot swarm, we designed a communication architecture based on local-only communication. This choice has the desirable properties of making attacks exploiting long-distance communications (more than 10 cm away) impossible and of being coherent with the self-organizing nature of robot swarms.

For the real-robot experiments, the communication architecture of the swarms consists of two layers. In the first layer, the robots maintain a Wi-Fi Mobile Ad-hoc Mesh Network (MANET) that allows them to establish TCP connections. This layer provides the bandwidth

required for the synchronization of blockchain information. As a MANET is a peer-to-peer network, it is decentralized and consistent with the requirements of swarm robotics research. However, due to its range of several meters, the MANET would facilitate remote attackers to interfere with the swarm. We have thus implemented a second layer that limits the range of the MANET by having robots reject TCP connections which are not in close vicinity, specifically not at a distance of 10 cm or less from each other (the robots detect the distance using their range-and-bearing module).

The robot controller

The controller of each robot is composed of five high-level routines, which are implemented using Python and the Ethereum blockchain software.

Random-walk with obstacle avoidance At each moment in time, a robot performs one of the following three basic behaviors: straight movement, rotation on site, or obstacle avoidance. If a robot's infrared sensors detect a close-by obstacle (< 5 cm), obstacle avoidance is selected in order to prevent collisions (the eight infrared sensors on the robot's frame are used to decide whether the robot should turn left or right to avoid the obstacle). Otherwise, the robot alternates between straight movement and rotation on site; the duration of each phase is sampled from an exponential distribution (*40*).

Estimation Once per second, a robot samples the floor via its ground sensor. This sample is then mapped to either black or white and is used to calculate a local estimate of the percentage of white tiles in the environment by dividing the number of white ground sensor readings by the total number of readings.

Peering Robots use their range-and-bearing actuators/sensors to simultaneously transmit their IDs and listen for other IDs within a range of approximately 10 cm. When an ID is received, the robot executes a TCP request to obtain the enode (a unique identifier of an Ethereum blockchain node, used for the peering calls) from the other robot. Once IR message exchanges no longer occur, the peer is removed and all local information regarding that peer is deleted.

Local estimate dissemination As soon as a robot has a sufficient number of crypto tokens (40 tokens in our experiments), it sends its local estimate to the smart contract, that uses them to generate a swarm estimate. Thus, the frequency of sent measurements is determined by the crypto-token economy.

Blockchain synchronization An instance of the Ethereum software (`geth`, version 1.8.0) is executed on each robot during the entire course of the experiment. In order to create new blocks on the blockchain, each robot acts as a sealer for the proof-of-authority consensus protocol and broadcasts the sealed blocks to its peers. When a block is received, the robot validates the contents of the block and updates its blockchain according to the specified consensus rules (see Supplementary Methods, "Proof-of-authority consensus algorithm").

The simulator

We conduct the simulations in the robot swarm simulator ARGoS (*44*), together with the ARGoS e-puck robot plugin (*50*), and the ARGoS-Python interface (*51*). To combine ARGoS and the smart contract framework Ethereum, we used the ARGoS-Blockchain interface (*33*) and extended it to make it compatible with Python (see Supplementary Methods, "Simulation framework: Installation and execution of the software"). This interface enables robots to act as blockchain nodes and provides them with access to blockchain functions. For the programming of the simulation framework we have paid attention to efficiency and parallelization; thus, it is possible to model swarms consisting of more than 100 robots.

In the simulations, we also implemented a two-layer communication architecture to replicate the communication capabilities of the real robots: the robots determine via their range-and-bearing module if they are 10 cm or closer to another robot. Whenever this is the case, TCP connections are established via the Docker network (see Supplementary Methods, "Simulation framework: Installation and execution of the software").

The simulations were executed on Amazon Elastic Compute Cloud (EC2). To simulate the limited hardware of the real Pi-puck robots, 1.0 GHz of CPU and 512 MB of RAM were assigned to each Docker container.

There is a close match between the results obtained using real robots and the results from the simulations (see Section Results). This outcome highlights the quality of our developed simulation environment—as the reality gap is small—and shows that it can be used for the prototyping, the development, and the study of different scenarios. Based on this validation, we complemented the experiments, which were executed both with real robots and in simulations, by extended studies in simulation only.

The experiment design

Initialization and Termination

The robots are randomly distributed in the arena at the start of each experimental run. In the real-robot experiments, this random arrangement is achieved by performing a random walk with

obstacle avoidance before starting an experimental run. In the simulations, the initial positions of the robots are drawn from the uniform distribution.

In addition, at the beginning of each experimental run, a new genesis block (the first block of a blockchain) is created and distributed among the robots. This genesis block contains the smart contract and the list of robots that are allowed to create new blocks in the proof-of-authority consensus protocol (in our experiments, all robots of the swarm).

An experimental run is stopped after all robots have received 'true' when querying the function `hasConverged()`.

Independent variables

Depending on the experiment, we vary one or more of the following independent variables: the swarm size, the number of Byzantine robots, the type of Byzantine fault, the dimension of the environment and the layout of the floor, and the convergence criterion. In the following, these variables are described in more detail.

Swarm size: Changing the swarm size, that is, the total number of robots in the swarm, allows for analyzing our approach in terms of two key features of robot swarms: scalability (the ability of the system to maintain or improve performance as the swarm size increases) and partition-tolerance (the ability of the system to reach consensus when there is reduced connectivity; in this case, induced by a more sparse distribution of the robots in the arena).

Number of Byzantine robots: To understand how well the presented approach can handle Byzantine robots, we vary their number in the swarm. When the number of Byzantines is increased, we keep the overall swarm size constant; that is, we do not add more robots to the swarm but change the ratio between the number of Byzantines and non-Byzantines. Keeping the swarm size constant ensures that other variables, such as the average connectivity, do not change. Given a certain number of Byzantine robots, which specific robots in an experiment are Byzantine and which are non-Byzantine is randomly determined by a script at the beginning of each experiment (both in reality and in simulation).

Type of Byzantine fault: By default, in our experiments, a Byzantine robot sends a faulty local estimate of '0 % white tiles' to the smart contract, independent of its actual ground sensor readings. This fault is well-motivated by our tests with physical robots and can occur both in laboratory environments as well as in real-world deployments—for example, in the following situations: a robot gets stuck on a black tile during the experiments because the wheels do not have enough grip and spin on the spot; a robot's ground sensor does not have the correct distance from the floor due to a loose screw; a software error occurs because of a crash of the communication protocol that causes the local estimates to remain at their initial value of 0; or

the robot is controlled by a malicious entity that sends extreme values in order to work against the goal of the swarm. We also consider two additional types of Byzantine faults: in the first new Byzantine fault, Byzantine robots send either 0 % or 100 % with an equal probability; in the second new fault, Byzantine robots send a value randomly drawn from the uniform distribution between 0 % and 100 %. The two additional types of Byzantine faults are motivated by the following situations: a Byzantine robot might have no knowledge about the environment (for example, because its ground sensor or its motor failed), therefore, the robot might try to "play the lottery" and send random estimates; a partially broken sensor, a broken connection, dust, or a broken software interface could introduce noise, which, without further information could be modeled by a uniform distribution or a Bernoulli distribution; or the robot is controlled by a malicious entity that tries to add noise to the swarm estimate.

Dimensions of the environment and layout of the floor: We use a $38 \times 38 = 1444$-tile arena as the default arena for our experiments. Each tile is either black or white and $5\,\mathrm{cm} \times 5\,\mathrm{cm}$ in size, resulting in a $3.61\,\mathrm{m}^2$ surface area. Additionally, we created smaller and larger arenas for Experiments 2b and 2c in which we keep the robot density constant across arenas of different size.

We chose the number of tiles per row (22, 31, 38, ...) in such a manner that it is possible to have arenas where the tile size remains constant and whose surface areas are n times as large as the surface area of the arena for 8 robots, where n is approximately an integer. In this way, we can have swarm sizes that are multiples of 8 while keeping a constant robot density (see Table 2).

In Experiment 2b, we used the three smaller arenas for runs with real robots and all the arena sizes in simulation. In Experiment 2c, which was run in simulation only, we used all the arena sizes.

All arenas have 25 % white tiles and 75 % black tiles. For all arena sizes, the layout of the floor (defined by the position of the tiles) was chosen uniformly at random using a Python script.

Table 2. Arena sizes for experiments with constant robot density (Experiments 2b and 2c).

Number of robots	Number of tiles	Arena size
8	$22 \times 22 = 484$	$1.10\,\mathrm{m} \times 1.10\,\mathrm{m} = 1.21\,\mathrm{m}^2$
16	$31 \times 31 = 961$	$1.55\,\mathrm{m} \times 1.55\,\mathrm{m} \approx 2.40\,\mathrm{m}^2$
24	$38 \times 38 = 1444$	$1.90\,\mathrm{m} \times 1.90\,\mathrm{m} = 3.61\,\mathrm{m}^2$
48	$54 \times 54 = 2916$	$2.70\,\mathrm{m} \times 2.70\,\mathrm{m} = 7.29\,\mathrm{m}^2$
72	$66 \times 66 = 4356$	$3.30\,\mathrm{m} \times 3.30\,\mathrm{m} = 10.89\,\mathrm{m}^2$
96	$76 \times 76 = 5776$	$3.80\,\mathrm{m} \times 3.80\,\mathrm{m} = 14.44\,\mathrm{m}^2$
120	$85 \times 85 = 7225$	$4.25\,\mathrm{m} \times 4.25\,\mathrm{m} \approx 18.06\,\mathrm{m}^2$

Once a layout was created, the same one was used for all experiments (with the exception of Experiment 1b – Changing tile layout, where a new layout is created for each run). For the real-robot experiments, we printed the floor layouts on thick paper.

Convergence criterion: By default, an experiment is stopped after all robots have received the convergence signal by querying the corresponding function of the smart contract. To evaluate the growth of the blockchain size over time, in Experiments 2c and 2d, the convergence signal is disabled, and the experiments are allowed to run for 600 minutes.

Performance measures

In order to evaluate the results, we use the following two performance measures: absolute error and convergence time. In Experiment 2c, we additionally measure the blockchain size, the inbound token flow, and the bandwidth utilization.

Absolute error: In all experiments, the goal of the swarm is to determine the fraction of white tiles. The swarm estimate is the collective result and aggregation of the local estimates of the individual robots, determined by the logic written in the smart contract. Since the purpose of the study is to determine whether the swarm reaches consensus on a value close to the correct one (25 % white tiles in all experiments), for each run, we extract the swarm estimate from only one robot (we use the last robot that received the convergence signal). If we were to extract the swarm estimates from all robots and aggregate them during our analysis, we would not necessarily test the swarm's ability to reach consensus: this aggregation could average out the variance in the swarm estimates and constitute an additional post-hoc consensus process.

The absolute error is the absolute difference between the swarm estimate and the actual fraction of white tiles. For example, if the swarm estimate is 27 %, the absolute error is $|25\,\% - 27\,\%| = 2\,\% = 0.02$.

In Figures 2, 4, and 5 we also plot the absolute error obtained with a very simple baseline algorithm. This baseline is calculated by averaging the individual estimates of the robots. The average is computed after a run is finished and does not use outlier detection or a blockchain. We could have chosen as a baseline the more sophisticated Byzantine linear consensus protocol (*38, 39*). However, as we have shown in (*33*), this algorithm is neither resistant to Sybil attacks nor has many of the desirable properties of our blockchain-based algorithm such as data logging, non-repudiation, and execution of decentralized programs.

Convergence time: We define the convergence time as the time it takes for all robots to receive the convergence signal from the smart contract. This convergence signal is set to `true` if the following two conditions are met: the absolute difference in the swarm estimate between the

previous proposal round and the current proposal round is smaller than $0.2\,\%$ and there were at least three proposal rounds. The second condition ensures that a minimum number of estimates are stored in the smart contract in order to increase the confidence in the collective estimate.

Blockchain size: To determine the blockchain size, we measure the size in MB of the Ethereum data folder of the last robot that received the convergence signal.

Inbound token flow: The inbound token flow of a robot is the cumulative sum of all the payments in crypto tokens (`UBI` payments and reward payments) it received during the experiment. Robots spend their crypto tokens again as soon as they have sufficient tokens to create a valid `sendEstimate(<localEstimate>)` transaction; therefore, analyzing the wealth instead of the token flow of the robots would be futile.

Bandwidth utilization: For each robot i, we record the total size S_i (in KB) of the sent messages, as well as the total time T_i a robot i is communicating with at least another robot (that is, the time a robot is able to exchange information). From this data, we can obtain the mean bandwidth utilization (that is, mean size of sent packets per second) both over the total runtime T of the experiment ($\sum_i^N (S_i/T)/N$), and over the time T_i when the robots are communicating ($\sum_i^N (S_i/T_i)/N$), where N is the number of robots in the swarm. Note that the total size of the sent packets equals the total size of the received packets since each sent packet is received by another robot, as it is a closed system without packet loss.

References

1. M. Brambilla, E. Ferrante, M. Birattari, M. Dorigo, Swarm robotics: A review from the swarm engineering perspective, *Swarm Intelligence* **7**, 1–41 (2013).

2. M. Dorigo, M. Birattari, M. Brambilla, Swarm robotics, *Scholarpedia* **9**, 1463 (2014).

3. L. Bayındır, A review of swarm robotics tasks, *Neurocomputing* **172**, 292–321 (2016).

4. H. Hamann, *Swarm Robotics: A Formal Approach* (Springer, Cham, Switzerland, 2018).

5. M. Dorigo, G. Theraulaz, V. Trianni, Reflections on the future of swarm robotics, *Science Robotics* **5**, abe4385 (2020).

6. M. Dorigo, G. Theraulaz, V. Trianni, Swarm robotics: Past, present and future, *Proceedings of the IEEE* **109**, 1152–1165 (2021).

7. G.-Z. Yang, *et al.*, The grand challenges of Science Robotics, *Science Robotics* **3** (2018).

8. F. Higgins, A. Tomlinson, K. M. Martin, Survey on security challenges for swarm robotics, *Proceedings of the Fifth International Conference on Autonomic and Autonomous Systems* (IEEE Press, 2009), pp. 307–312.

9. E. Bonabeau, M. Dorigo, G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems* (Oxford University Press, New York, 1999).

10. M. Dorigo, *et al.*, Evolving self-organizing behaviors for a swarm-bot, *Autonomous Robots* **17**, 223–245 (2004).

11. A. F. T. Winfield, J. Nembrini, Safety in numbers: Fault tolerance in robot swarms, *International Journal on Modelling Identification and Control* **1**, 30–37 (2006).

12. A. L. Christensen, R. O'Grady, M. Birattari, M. Dorigo, Fault detection in autonomous robots based on fault injection and learning, *Autonomous Robots* **24**, 49–67 (2008).

13. A. L. Christensen, R. O'Grady, M. Dorigo, From fireflies to fault-tolerant swarms of robots, *IEEE Transactions on Evolutionary Computation* **13**, 754–766 (2009).

14. J. D. Bjerknes, A. F. T. Winfield, On fault tolerance and scalability of swarm robotic systems, *The 10th International Symposium on Distributed Autonomous Robotic Systems (DARS 2013)*, A. Martinoli, *et al.*, eds. (Springer, Berlin/Heidelberg, Germany, 2013), pp. 431–444.

15. D. Tarapore, A. L. Christensen, J. Timmis, Generic, scalable and decentralized fault detection for robot swarms, *PLOS ONE* **12**, 1-29 (2017).

16. D. Tarapore, P. U. Lima, J. Carneiro, A. L. Christensen, To err is robotic, to tolerate immunological: fault detection in multirobot systems, *Bioinspiration & Biomimetics* **10**, 016014 (2015).

17. J. O'Keeffe, D. Tarapore, A. G. Millard, J. Timmis, Adaptive online fault diagnosis in autonomous robot swarms, *Frontiers in Robotics and AI* **5**, 131 (2018).

18. D. Tarapore, J. Timmis, A. L. Christensen, Fault detection in a swarm of physical robots based on behavioral outlier detection, *IEEE Transactions on Robotics* **35**, 1516-1522 (2019).

19. V. Strobel, E. Castelló Ferrer, M. Dorigo, Managing Byzantine robots via blockchain technology in a swarm robotics collective decision making scenario, *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2018)*, M. Dastani, G. Sukthankar, E. André, S. Koenig, eds. (International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, USA, 2018), pp. 541–549.

20. A. Aswale, A. López, A. Ammartayakun, C. Pinciroli, Hacking the colony: On the disruptive effect of misleading pheromone and how to defend against it, *Proceedings of the 21st International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2022)*, P. Faliszewski, V. Mascardi, C. Pelachaud, M. E. Taylor, eds. (International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, USA, 2022), pp. 27–34.

21. E. Castelló Ferrer, T. Hardjono, A. Pentland, M. Dorigo, Secure and secret cooperation in robot swarms, *Science Robotics* **6**, abf1538 (2021).

22. G. Primiero, E. Tuci, J. Tagliabue, E. Ferrante, Swarm attack: A self-organized model to recover from malicious communication manipulation in a swarm of simple simulated agents, *Swarm Intelligence – Proceedings of ANTS 2018 – Eleventh International Conference*, M. Dorigo, *et al.*, eds. (Springer, Cham, Switzerland, 2018), pp. 213–224.

23. G. Maître, E. Tuci, E. Ferrante, Opinion dissemination in a swarm of simulated robots with stubborn agents: A comparative study, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2020)* (IEEE Press, Piscataway, NJ, USA, 2020), pp. 1–6.

24. I. Sargeant, A. Tomlinson, Maliciously manipulating a robotic swarm, *Proceedings of ESCS'16 – The 14th International Conference on Embedded Systems, Cyber-physical Systems, & Applications* (CSREA Press, 2016), pp. 122–128.

25. E. R. Hunt, S. Hauert, A checklist for safe robot swarms, *Nature Machine Intelligence* **2**, 420–422 (2020).

26. L. Lamport, R. Shostak, M. Pease, The Byzantine generals problem, *ACM Transactions on Programming Languages and Systems (TOPLAS)* **4**, 382–401 (1982).

27. M. Castro, B. Liskov, Practical Byzantine fault tolerance and proactive recovery, *ACM Transactions on Computer Systems* **20**, 398—461 (2002).

28. S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, *Tech. rep.* (2008). `https://bitcoin.org/bitcoin.pdf`, Accessed Aug. 11, 2018.

29. V. Buterin, A next-generation smart contract and decentralized application platform. Ethereum project white paper., *Tech. rep.*, Ethereum Foundation (2014). `https://github.com/ethereum/wiki/wiki/White-Paper`, Accessed Jul. 18, 2019.

30. E. Castelló Ferrer, The blockchain: A new framework for robotic swarm systems, *e-print* (2016). ArXiv:1608.00695v3.

31. V. Strobel, M. Dorigo, Blockchain technology for robot swarms: A shared knowledge and reputation management system for collective estimation, *Swarm Intelligence – Proceedings of ANTS 2018 – Eleventh International Conference*, M. Dorigo, *et al.*, eds. (Springer, Cham, Switzerland, 2018), vol. 11172 of *Lecture Notes in Computer Science*, pp. 425–426.

32. J. Peña Queralta, L. Qingqing, Z. Zou, T. Westerlund, Enhancing autonomy with blockchain and multi-access edge computing in distributed robotic systems, *Proceedings of the 5th International Conference on Fog and Mobile Edge Computing (FMEC 2020)* (IEEE Press, Piscataway, NJ, USA, 2020), pp. 180–187.

33. V. Strobel, E. Castelló Ferrer, M. Dorigo, Blockchain technology secures robot swarms: A comparison of consensus protocols and their resilience to Byzantine robots, *Frontiers in Robotics and AI* **7**, 54 (2020).

34. A. Reina, Robot teams stay safe with blockchains, *Nature Machine Intelligence* **2**, 240–241 (2020).

35. A. Pacheco, V. Strobel, A. Reina, M. Dorigo, Real-time coordination of a foraging robot swarm using blockchain smart contracts, *Swarm Intelligence – Proceedings of ANTS 2022 – Thirteenth International Conference* (Springer, Berlin, Germany, 2022), vol. 13491 of *Lecture Notes in Computer Science*, pp. 196–208.

36. M. G. Santos De Campos, C. P. Chanel, C. Chauffaut, J. Lacan, Towards a blockchain-based multi-UAV surveillance system, *Frontiers in Robotics and AI* **8**, 557692 (2021).

37. J. Grey, O. Seneviratne, I. Godage, Blockchain-based mechanism for robotic cooperation through incentives: Prototype application in warehouse automation, *Proceedings of the 2021 IEEE International Conference on Blockchain (Blockchain 2021)* (IEEE Press, Piscataway, NJ, USA, 2021), pp. 597–604.

38. L. Guerrero-Bonilla, A. Prorok, V. Kumar, Formations for resilient robot teams, *IEEE Robotics and Automation Letters* **2**, 841–848 (2017).

39. D. Saldaña, A. Prorok, S. Sundaram, M. F. M. Campos, V. Kumar, Resilient consensus for time-varying networks of dynamic agents, *Proceedings of the American Control Conference (ACC)* (IEEE Press, 2017), pp. 252–258.

40. G. Valentini, D. Brambilla, H. Hamann, M. Dorigo, Collective perception of environmental features in a robot swarm, *Swarm Intelligence – Proceedings of ANTS 2016 – Tenth International Conference*, M. Dorigo, *et al.*, eds. (Springer, Cham, Switzerland, 2016), vol. 9882 of *Lecture Notes in Computer Science*, pp. 65–76.

41. G.-T. Nguyen, K. Kim, A survey about consensus algorithms used in blockchain, *Journal of Information Processing Systems* **14**, 101–128 (2018).

42. A. G. Millard, *et al.*, The Pi-puck extension board: A Raspberry Pi interface for the e-puck robot platform, *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE Press, 2017), pp. 741–748.

43. F. Mondada, *et al.*, The e-puck, a robot designed for education in engineering, *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, P. J. S. Gonçalves, P. J. D. Torres, C. M. O. Alves, eds. (Instituto Politécnico de Castelo Branco, 2009), pp. 59–65.

44. C. Pinciroli, *et al.*, ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems, *Swarm Intelligence* **6**, 271–295 (2012). Version 3.0.0-beta48.

45. E. B. Hamida, K. L. Brousmiche, H. Levard, E. Thea, Blockchain for enterprise: Overview, opportunities and challenges, *The Thirteenth International Conference on Wireless and Mobile Communications (ICWMC 2017)* (2017), pp. 83–88.

46. P. Szilágyi, EIP 225: Clique proof-of-authority consensus protocol (2017). Accessed May 10, 2020.

47. E. Drasutis, Iota smart contracts, *Tech. rep.*, IOTA Foundation (2021). https://files.iota.org/papers/ISC_WP_Nov_10_2021.pdf, Accessed Apr. 13, 2023.

48. J. Polge, J. Robert, Y. Le Traon, Permissioned blockchain frameworks in the industry: A comparison, *ICT Express* **7**, 229–233 (2021).

49. H. Pervez, M. Muneeb, M. U. Irfan, I. U. Haq, A comparative analysis of DAG-based blockchain architectures, *Proceedings of the 12th International Conference on Open Source Systems and Technologies (ICOSST 2018)* (IEEE Press, Piscataway, NJ, USA, 2018), pp. 27–34.

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

51. K. Hasselmann, A. Parravicini, A. Pacheco, V. Strobel, Python wrapper for ARGoS 3 simulator, https://github.com/KenN7/argos-python/ (2021).

52. V. Buterin, V. Griffith, Casper the friendly finality gadget, *e-print* (2017). ArXiv:1710.09437v4.

53. A. Pacheco, V. Strobel, M. Dorigo, A blockchain-controlled physical robot swarm communicating via an ad-hoc network, *Swarm Intelligence – Proceedings of ANTS 2020 – Twelth International Conference*, M. Dorigo, *et al.*, eds. (Springer, Cham, Switzerland, 2020), vol. 12421 of *LNCS*, pp. 3–15.

54. D. Merkel, Docker: Lightweight linux containers for consistent development and deployment, *Linux J.* **2014** (2014).

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

Acknowledgments

We thank Mary Katherine Heinrich for providing useful comments and for proof-reading this manuscript.

Authors contributions

All authors contributed to the conceptualization of this research and to the setup of the experiments. A.P. and V.S. implemented the robots' controllers and the experimental setup. V.S. ex-

ecuted the experiments and analyzed the results. The manuscript was written by V.S. and M.D. and revised by all authors.

Competing interests

The authors declare that they do not have any competing interests.

Data and materials availability

All collected data is available as CSV files on Dryad under the DOI:
`doi:10.5061/dryad.zcrjdfnj3`
Contact V.S. at `volker.strobel@ulb.be` for any questions. The real-robot experiments are recorded on videos that can be requested from the authors.

Funding

This work was partially supported by the Program of Concerted Research Actions (ARC) of the Université Libre de Bruxelles. V.S. and M.D. acknowledge support from the Belgian F.R.S.-FNRS, of which they are a Research Fellow and a Research Director respectively. V.S. also gratefully acknowledges support from the Fondation Jaumotte-Demoulin and the Fonds David et Alice Van Buuren. A.P. acknowledges support via a fellowship from the Faculty of Applied Sciences of the Université Libre de Bruxelles.

Supplementary Materials for

Robot swarms neutralize harmful Byzantine robots using a blockchain-based token economy

Volker Strobel,[1*] Alexandre Pacheco,[1] Marco Dorigo[1]

[1]IRIDIA, Université Libre de Bruxelles, Brussels, Belgium

[*]To whom correspondence should be addressed; E-mail: volker.strobel@ulb.be.

The PDF file includes:

Supplementary Results

Rate of estimates

Figure S1 shows the rate of estimates in a period of 600 minutes for both non-Byzantine and Byzantine robots. The number of estimates sent by non-Byzantine robots grows over time because of the UBI. This growth is logarithmic because of the exponential scheme of the UBI payments (the UBI is paid in blocks $1, 2, 4, 8, ...$). By contrast, the Byzantine robots send most of their estimates at the beginning of the experiment when the UBI is more frequent. All these UBI payments received by the Byzantine robots are redistributed to the non-Byzantines by the reward mechanism. The Byzantine robots continue to send estimates with an exponentially decreasing frequency, because of the exponential scheme of the UBI payments they continue to receive.

CPU and RAM utilization

In order to investigate the blockchain overhead, we analyzed the actual CPU and RAM utilization in a preliminary study using real Pi-puck robots. To do so, we implemented a Pi-puck baseline controller that does not execute the Ethereum blockchain software; in the baseline controller no blockchain interactions (such as sending transactions or querying the state of variables) are active. Using the baseline controller, we observed that the mean CPU utilization was 5.3 % and the mean RAM utilization was 24.7 %. The utilization of the system resources did not grow over time using the baseline controller.

To determine the blockchain overhead (that is, the utilization of the system resources when executing the Ethereum software compared to the baseline), we executed a 60-minute run with 8 robots using our blockchain-based controller. We observed that the CPU utilization was largely independent of time, with a mean of 31.3 %. The RAM utilization grew over time, from 35.0 % in the beginning of the experiment to 40.5 % after 60 minutes. Therefore, in comparison to the baseline controller, we can estimate that the blockchain software occupied a mean of 26.0 % of the available CPU resources and a maximum of 15.8 % of the available RAM resources in the 60-minute run.

We then investigated in simulation whether or not the RAM utilization continues to grow after 60 minutes. We observed that in the first 60 minutes the RAM utilization was a close match to what observed with the real robots. In the following 120 minutes it grew up to 45 % and from 180 minutes to the end of the experiment (600 minutes) it stabilized at 45 %. The stabilization is due to the fact that Ethereum is implemented to only hold a maximum number of blocks in memory (see `https://geth.ethereum.org/docs/interface/sync-modes`).

Even though these results give a clear initial indication that our system can be used on real

robots without a major effect on resource utilization, we acknowledge that further studies will be needed to fully assess the system resource utilization of blockchain technology.

Supplementary Discussion

More sophisticated Byzantine behavior

As we discussed in Section Discussion – Challenges, the smart contract that we have designed is rather simple and could be circumvented by smarter Byzantine robots. The range of possible Byzantine behaviors and how smart contracts could handle them will have to be studied in more details in future research, both theoretically and empirically. Here we discuss two possible attacks and how our blockchain-based robot swarm could neutralize them using extensions of our current smart contract.

Eavesdropping. This is an attack where a robot uses knowledge about estimates sent by other robots in order not to send outliers. However, the disruption to the global estimate created by this type of attack is likely limited since in our smart contract such an estimate has to be within ± 0.2 from the current swarm estimate (otherwise the estimate would be discarded). This limited disruption is further underpinned by the fact that we assume that the Byzantine robots are in the minority (which is a general prerequisite for consensus achievement in decentralized systems and distributed ledger technology usage in general); therefore, there is a high probability that the non-Byzantines estimates would outweigh the Byzantine estimates.

There is however a low probability that the Byzantine robots succeed and slowly bias the estimate in one direction. To prevent this, using knowledge about other robots' estimates could be made impossible. For example, the robots could first send only a commitment for the estimate using the estimate hash value rather than the estimate itself. Only when the smart contract has received enough commitments, the robots are asked to reveal the actual estimates. The smart contract can then check if the sent hash values in the commitment phase match the hash value of the revealed estimates, so that the robots cannot change the values they committed to.

Another approach would be to use a different blockchain framework with privacy-preserving features. For example, the Secret Network (see `https://scrt.network/`) precludes eavesdropping attacks using "secret smart contracts" that allow for calculations on encrypted data; in our case, they could be used to compute the mean of the encrypted estimates.

Delayed Byzantine behavior. Inherent to reputation management systems is the issue of "delayed Byzantine behavior," an attack where a robot first acts according to the protocol and later on becomes Byzantine. We believe that our system copes with this issue to a satisfying degree, as explained in the following. In order to perform such a delayed attack, a Byzantine robot would first need to accumulate sufficient crypto tokens. There are only two ways to collect

crypto tokens: via the UBI and via rewards. Because the UBI alone is not sufficient as the interval between two payments increases exponentially with time, the Byzantine robot needs to receive rewards and the only way to do so is by sending "inliers," that is, estimates that are not rejected by the smart contract. However, even once it has accumulated enough crypto tokens, the Byzantine robot can only disrupt the systems by sending "almost" outliers (that is, estimates that, although within the range of acceptability for the smart contract, situate themselves at an extreme of the range) and try to slowly steer the swarm estimate in one direction, transforming this attack into the eavesdropping already discussed above.

Supplementary Methods

Proof-of-authority consensus algorithm

The Ethereum blockchain protocol lets a user choose among three consensus algorithms: proof-of-work, proof-of-stake, and proof-of-authority. These algorithms incentivize the production of valid blocks by exposing block producers to risk when they generate invalid blocks: in proof-of-work block producers risk wasting computational resources and energy in case of producing invalid blocks; in proof-of-stake (*52*) they risk losing monetary stakes; and in proof-of-authority (*46*) there is a list of authorized block producers who risk losing that privilege if they are voted out for being dishonest.

Robot swarms have limited resources and battery life, therefore, we argue that both proof-of-stake and proof-of-authority are suitable and energy-efficient choices. Since proof-of-stake was still in a very early stage of development when we started the present research, we decided to select proof-of-authority. As opposed to proof-of-work where a computationally expensive mining mechanism is necessary to achieve consensus, proof-of-authority is computationally lightweight as it only requires a majority of preselected nodes (in this work, a majority of robots) to agree on the state of the blockchain database; and in contrast to proof-of-stake, proof-of-authority has the advantage of allowing for consensus where all agents participate with equal rights (which in this case, are delegated to the robots by the swarm designers). If proof-of-stake with equal stakes was to be used, the expected behavior would be the same as the one observed with proof-of-authority.

In the proof-of-authority protocol, there are two kinds of blockchain nodes: regular blockchain nodes and sealer nodes. The latter play the same role as miners in proof-of-work and are able to add blocks to the blockchain by signing them. In order to solve conflicting situations, the nodes agree on the strongest chain, that is the chain with the highest difficulty. The difficulty is calculated as follows. There is a preferred sealer for each block, chosen in a round-robin fashion. If the preferred sealer signs the block, it is called an in-turn signature; however, if the

preferred sealer cannot sign (for example because it is unreachable or it is reached with a large delay) then another sealer signs it with a so-called an out-of-turn signature. The difficulty for in-turn signature is 2 and the one for out-of-turn signature is 1. Since the local communication in our setup leads to delays in the propagation of blockchain information across the robot swarm, most signatures in our experiments are out-of-turn signatures. N sealers are initially specified in the genesis block; sealers can later be added or removed based on majority voting. The sealers can sign new blocks and disseminate them in the network at any time. However, in order for a block to be valid and accepted by the other nodes in the network, each receiver of the new block checks whether the following conditions are met:

- the timestamp of the new block must be at least $t = 15$ seconds after the previous block (also known as the block time);
- a sealer can only sign one block in $\lfloor \frac{N}{2} \rfloor + 1$ blocks (to guarantee majority consensus);
- a sealer must create a correct signature using its private key and sign the hash of the current block.

Although in this work the size of our swarms does not change during an experiment and all robots belong to a same owner, we expect proof-of-authority to also work with swarms in which the number of robots changes at runtime and does not have any requirement on the fact that robots belong or not to a same owner. In practice, this could be implemented as follows:

1. In the initial swarm every robot is a sealer.
2. Later added robots join without sealer privileges.
3. The universal basic income (UBI) is assigned only to sealers.
4. Newly added non-sealer robots initially get some crypto tokens via a vouching system: existing sealers can give a part of their crypto tokens to the new robots; in return, they get a share of the crypto tokens when the newly added non-sealer robots get a reward. If the newly added non-sealer robots are Byzantine, they lose their crypto tokens—which are redistributed to the non-Byzantine robots via the reward mechanism—and can no longer disrupt the swarm.

Simulation framework: Installation and execution of the software

Here we describe the setup of the simulation framework which is based on the following modules:

- The ARGoS robot swarm simulator (*44*). This physics-based simulator allows for executing experiments with large robot swarms in real-time.
- The ARGoS-Blockchain interface (Blockchain module) (*33, 53*). This interface allows for creating a custom Ethereum network with a specified number of nodes, where each node (each robot in our work) is located in a separate Docker container (*54*). Our simulated robots thus maintain an Ethereum blockchain network and can exchange blockchain information via the Docker network infrastructure. We enhanced the Blockchain module with the feature to create a proof-of-authority network. A corresponding genesis block is created that specifies the sealers. Furthermore, in the extension, the smart contracts are inserted into the genesis block. Thus, at the beginning of the experiment, the blockchain is immediately up and running for the desired mission.
- The ARGoS-Python interface (*51*). This interface makes it possible to write ARGoS controllers and loop functions in Python instead of C++. For this research, we extended the interface to make it compatible with ARGoS loop functions and the Ethereum software. In order to execute an Ethereum function (such as sending a transaction) from ARGoS, we implemented the following architecture:
  - The robots' controllers are written in Python using the ARGoS-Python interface.
  - To communicate with the Ethereum processes in the Docker containers, we use `Web3.py` (see `https://web3py.readthedocs.io`). This collection of libraries allows for connecting to the Ethereum processes using the WebSocket communication protocol.
  - To make `Web3.py` accessible outside of the Docker containers, each Docker container hosts an `RPyC` server (see `https://rpyc.readthedocs.io`); each ARGoS controller connects to this server to interact with the Ethereum process of a particular robot.
  - The communication channels between the Docker containers are only established when robots are within a 10-cm communication range in order to simulate the local communication capabilities of real robots.

The following setup steps describe the installation of the ARGoS-Blockchain interface using Debian Linux as the operating system.

Installation of ARGoS

Here we briefly describe the setup of ARGoS. More information is given in the README file of the repository.

1. Install the dependencies.

```
$ sudo apt-get install git build-essential cmake g++ libfreeimage-dev \
  libfreeimageplus-dev qt5-default freeglut3-dev libxi-dev libxmu-dev \
  liblua5.3-dev lua5.3 doxygen graphviz graphviz-dev asciidoc
```

2. Obtain the source code.

```
$ git clone https://github.com/ilpincy/argos3.git
```

3. Build and install ARGoS system-wide.

```
$ cd argos3/
$ mkdir build
$ cd build
$ cmake ../src
$ make -j4
$ make doc
$ sudo make install
$ sudo ldconfig
```

4. Verify that the installation was successful.

```
$ argos3 --version
ARGOS_VERSION=3.0.0-beta59
ARGOS_INSTALL_PREFIX=/usr/local
ARGOS_USE_DOUBLE=ON
ARGOS_WITH_LUA=ON
ARGOS_BUILD_FLAGS= -Wall -g -ggdb3
```

Installation of the ARGoS e-puck plugin

Here we briefly describe the setup of the ARGoS e-puck plugin. More information is given in the README file of the repository.

1. Obtain the source code.

```
$ cd ~
$ git clone https://github.com/demiurge-project/argos3-epuck.git
```

2. Build and install the ARGoS-epuck plugin.

```
$ cd argos3-epuck/
$ mkdir build
$ cd build
$ cmake ../src
$ make
$ sudo make install
$ sudo ldconfig
```

Installation of Docker

The installation of the Docker Engine on Ubuntu is described at `https://docs.docker.com/engine/install/ubuntu/`. Here, we only describe how to control the Docker Engine as a non-root user.

1. By default, the execution of Docker commands requires root user privileges. To facilitate the execution of Docker commands, it is possible to run Docker as a non-root user (that is, also without using `sudo`) as follows (see `https://docs.docker.com/engine/install/linux-postinstall/` for more detailed instructions).

```
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
$ newgrp docker
```

2. Additionally, restarting the Docker engine via `systemctl` requires sudo privileges. To control Docker as a normal user, add an entry in the sudoers file (replace `<user>` with your user name).

```
$ sudo visudo -f /etc/sudoers.d/docker

<user> ALL= NOPASSWD: /bin/systemctl restart docker.service
<user> ALL= NOPASSWD: /bin/systemctl start docker.service
<user> ALL= NOPASSWD: /bin/systemctl stop docker.service
```

Installation of the simulation framework

1. First, the dependencies have to be installed.

```
$ sudo apt-get install g++ cmake git libboost-python-dev
```

2. Then, the source code of our simulation framework must be downloaded.

```
$ git clone --recurse-submodules \
  https://github.com/iridia-ulb/blockchain-simulations.git
```

   The source code of the simulation framework consists of the following folders:

```
blockchain-simulations/
├── FloorEstimation/ ..... Experiment folder
├── argos-blockchain/ ..... Docker blockchain module
└── argos-python/ ..... ARGoS-Python interface
```

3. After downloading, the ARGoS-Python interface has to be compiled.

```
$ cd blockchain-simulations/argos-python
$ git fetch
$ git checkout temp
$ mkdir build
$ cd build
$ cmake ..
$ make
```

4. Then the Docker image must be created. The image contains the Ethereum software and other scripts that simplify the interaction with Ethereum. Furthermore, the Docker swarm must be initialized. A Docker swarm provides the ability to launch many Docker containers in parallel and allows for assigning limits to each container, such as how much CPU and RAM it can consume.

```
$ cd argos-blockchain/geth/
$ docker build -t mygeth .
$ docker swarm init
```

5. Our smart contracts are written in the Solidity programming language. For this reason, `solc`, a compiler for Solidity, must be installed.

```
sudo add-apt-repository ppa:ethereum/ethereum
sudo apt-get update
sudo apt-get install solc
```

6. The Python controllers require additional packages.

```
$ sudo apt install python3-pip
$ pip3 install rpyc psutil
```

7. Optional: Experiment 1b uses a dynamic floor layout. The following steps are required to be able to create new floor layouts at the start of each experiment.

```
$ sudo apt install python3-pip
$ pip3 install numpy
$ pip3 install matplotlib
$ sudo apt install imagemagick
```

Setup of variables

In the following steps, we set variables so that the different modules are connected with each other.

1. First, the blockchain module needs to be linked to the parent folder that contains the different modules. For this purpose, the variable `MAINFOLDER` in the file `global_config_blockchain.sh` needs to be edited to point to the parent folder.

```
$ vim argos-blockchain/local_scripts/global_config_blockchain.sh
  MAINFOLDER="$HOME/blockchain-simulations"
```

2. Subsequently, the experiment folder must also know where the parent folder is located. Edit the variable `MAINFOLDER` in the file `experimentconfig.sh`:

```
$ vim blockchain-simulations/FloorEstimation/experimentconfig.sh
  export MAINFOLDER="$HOME/blockchain-simulations"
```

Run

An experiment is always started from the experiment base folder.

```
$ cd ~/blockchain-simulations/FloorEstimation/
$ source starters/G1/starter.sh # Start Experiment 1
```

It is also possible to start the Ethereum network without ARGoS, using the following command:

```
$ bash local_scripts/start_network.sh <number of nodes>
```

For example, `bash local_scripts/start_network.sh 5`, creates a custom Ethereum network with five nodes.

Physical Pi-puck robots: Installation and execution of the software

The experiments are started from a host which must possess wireless LAN. Before the start of an experiment, this computer joins the ad-hoc network of the robots. It then determines which robots are currently active, and creates and distributes a genesis block based on this information. The genesis block also includes the smart contract for an experiment. During an experiment, the host does not interact with the robots or the network. After an experiment is finished (indicated by the robots' green LEDs), one can retrieve the information that is stored on each robot (such as blockchain information, local estimates, and blockchain size) via the host. In addition, in the research and development phase, the host is able to update the robots' control software.

Installation on robots

The installation of blockchain software on the Pi-puck robots is described in a separate technical report (*55*). This technical report also describes how to establish an ad-hoc network among Pi-puck robots.

Installation on host

- Obtain source code:

```
$ git clone \
  https://github.com/iridia-ulb/blockchain-pipucks.git
```

- Indicate base folder:

```
$ cd blockchain-pipucks/
$ vim globalconfig
  # point to the folder blockchain-pipucks/
  export MAIN_DIR="/home/$USER/$BASE_FOLDER"
```

- Set up ad-hoc network:

```
$ vim setup-adhoc
  IF=<network interface name>
```

Quickstart

First, all robots that should be part of an experiment need to be turned on. The robots then automatically join the ad-hoc network and are ready to receive commands. Then, the following commands need to be executed on the host:

```
$ cd blockchain-pipucks/
$ ./setup-adhoc # Setup adhoc network on the host
$ cd control/monitor-pc/
$ ./ping-all # Collect robot IDs
$ ./scp-all sshkeys  # Send SSH keys to robots
$ ./scp-all all # Send control scripts
$ ./calibrate-gs # Calibrate the ground sensors
$ ./reset-all # Reset blockchain
```

The robots are then ready to start an experiment.

The following command, executed on the host, starts an experiment with N robots of which k are Byzantine robots.

```
$ ./start-experiment <N> <k> --rebuild
```

The flag `--rebuild` creates a new genesis block and compiles the smart contract. Therefore, it is only needed if the number of robots or the used robots change.

After a consensus has been reached, which is signaled by the robots' green LEDs, the log files can be stored in the folder `<dir>` on the host as follows.

```
$ ./collect-logs <dir>
```

Example:

```
$ ./start-experiment 10 2 --rebuild
# Wait for experiment to finish
$ ./collect-logs 10rob-02byz-01
```

This command starts an experiment with 10 robots of which 2 are Byzantine robots. The log files are then stored in the folder `10rob-02byz-01/`.

Detailed explanation of functions

```
setup-adhoc
```

This script initializes the ad-hoc network on the host. This is accomplished by setting the selected Wi-Fi device on the host to the Independent Basic Service Set (IBSS) mode.

```
ping-all [OPTION]
```

This script pings all robots by iterating over possible IP addresses. The responding robots are then added to a list. This guarantees that the host knows about the robots that are taking part in an experiment. Options:

`--slow`
Waits between pings in order to prevent network congestion.

`--signal`
Lets robots flash their LEDs in order to identify non-responding robots.

```
scp-all [OPTION]
```

This script uploads the specified files to the robots. It must be executed every time a file used by robots is changed on the host. Options:

`mainloop`
Send main control loop.

`gs`
Send ground sensor module.

erb
Send range-and-bearing module.

scs
Compile smart contract and send the ABI and binary files.

control
Send all control files.

setup-adhoc
Send adhoc setup file.

setup-geth
Send geth setup file.

all
Deletes and restores the robots' main directory (unlike the other commands, this command deletes the sub-folders of `blockchain-pipucks/` and uploads them again to the robots).

```
calibrate-gs
```

The function `calibrate-gs` calibrates the robots' ground sensors. To this end, this script makes the robots collect data for a specified amount of time (default: 3 minutes), then collects this data from the robots, computes thresholds by identifying a bi-modal distribution, and finally sends the computed thresholds back to the robots.

```
collect-logs <dir>
```

This function needs to be executed at the end of each experimental run. The script queries each robot for the data logs and transfers them via the Secure Copy Protocol (SCP) to the folder `<dir>` on the host.

```
i2cdetect-all
```

This script prints the battery status and the I^2C status of each robot. A broken, flat, or poorly connected battery can lead to a (temporary) failure of the I^2C connection. This failure in turn can lead to the situation in which the robot resets its range-and-bearing-module distance and communicates even with robots that are farther away than the 10 cm range we set for our experiments.

```
tmux-all [OPTIONS]
```

Opens a `tmux` pane for each robot in order to send commands. By default, it opens a bash terminal in the home folders of the robots. Options:

`geth`
Open bash terminal in the geth folder and waits for Enter to start an experiment.

`mainloop`
Open bash terminal in the mainloop folder and waits for Enter to start an experiment.

```
reset-keys
```

This function needs to be executed each time that new robots are used for the first time and the SSH keys need to be distributed. The function deletes the key files and sends the correct key to the specified robots.

```
reset-geth
```

This function needs to be executed every time a new experiment is started. It deletes the Ethereum folder and restarts the Ethereum client. It is used before a new experiment if the robot number and IDs stay the same. If new robots are used, `reset-all` is needed.

```
reset-genesis
```

This function recompiles the genesis block and uploads it to the robots. It is needed when the swarm size changes (since the genesis block specifies who is allowed to create a new block in the proof-of-authority consensus protocol), or when the smart contract needs to be recompiled.

```
reset-all
```

This function resets `geth`, creates a new genesis block, and redistributes the SSH keys. This function is required when a new experiment is started and one or more of the following parameters change: the swarm size, the robots in use, or the smart contract.

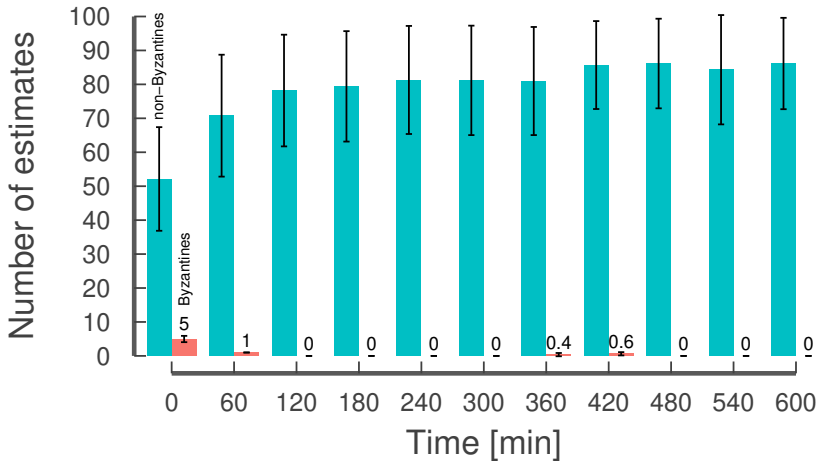

Fig. S1. Rate of estimates. Each bar represents the mean number of estimates sent by each robot within 60 minutes (cyan bars: non-Byzantine robots; red bars and numbers at the bottom of the plot: Byzantine robots). The error bars represent the standard deviation (data collected from 20 runs using $N = 24$ robots per run). The rate of estimates of non-Byzantine robots increases over time as they receive more and more crypto tokens thanks to the UBI and reward payments. The Byzantine robots send most of their estimates at the beginning of the experiment when the UBI is more frequent. Subsequently, they can only send estimates when they receive a UBI payment, as they do not get any reward payments.