This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3257984

Partition-Tolerant and Byzantine-Tolerant Decision-Making for Distributed Robotic Systems with IOTA and ROS 2

Farhad Keramat, Jorge Peña Queralta, and Tomi Westerlund

***Abstract*—With the increasing ubiquity of autonomous robotic solutions, the interest in their connectivity and in the cooperation within multi-robot systems is rising. Two aspects that are a matter of current research are robot security and secure multi-robot collaboration robust to byzantine agents. Blockchain and other distributed ledger technologies (DLTs) have been proposed to address the challenges in both domains. Nonetheless, some key challenges include scalability and deployment within real-world networks. This paper presents an approach to integrating IOTA and ROS 2 for more scalable DLT-based robotic systems while allowing for network partition tolerance after deployment. This is, to the best of our knowledge, the first implementation of IOTA smart contracts for robotic systems, and the first integrated design with ROS 2. This is in comparison to the vast majority of the literature which relies on Ethereum. We present a general IOTA+ROS 2 architecture leading to partition-tolerant decision-making processes that also inherit byzantine tolerance properties from the embedded blockchain structures. We demonstrate the effectiveness of the proposed framework for a cooperative mapping application in a system with intermittent network connectivity. We show both superior performance with respect to Ethereum in the presence of network partitions, and a low impact in terms of computational resource utilization. These results open the path for wider integration of blockchain solutions in distributed robotic systems with less stringent connectivity and computational requirements.**

***Index Terms*—DLT; Multi-robot systems; IOTA; Smart contracts; Blockchain; Ethereum; Cooperative mapping;**

I. Introduction

Autonomous robots are revolutionizing industries and civil applications. Two aspects that are part of today's ubiquitous robotic solutions are connectivity and teaming [1], [2], [3]. Indeed, many robots today are deployed as part of larger fleets or in teams, often heterogeneous and fruit of the combination of robots from different vendors. In addition, the proliferation of robotic systems is leading to increased connectivity and reliance on multi-robot interaction or cloud-based services [4]. Maintaining security becomes more critical as robots become more connected and ubiquitous. Additionally, multi-robot systems may be prone to malicious behavior from within the system, while malfunctioning units or sensors can also lead to unexpected functionality or results. In practical applications, a key aspect and important part of a multi-robot systems is collective decision-making [5], which is highly susceptible to malicious behavior. However, the potential effects of byzantine agents has not been always considered in the literature [6].

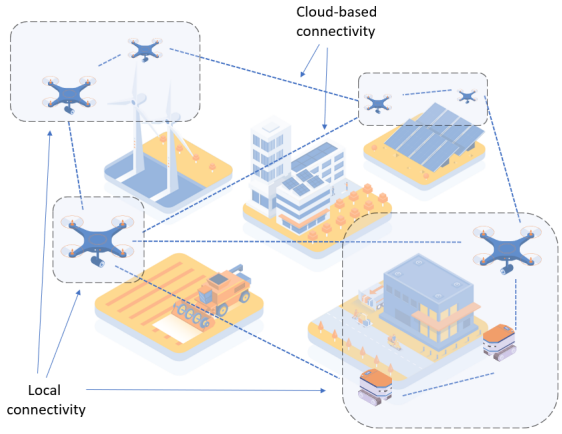


Fig. 1: Large-scale deployments of connected robots often include both cloud-based connectivity and local connectivity within subsets of robots, leading to potential network partitions. Global peer-to-peer connection cannot be always relied upon when deployments occur in remote areas. This is particularly critical when considering traditional blockchain-based solutions in which data is lost if a subset of robots are disconnected for a period of time.

In general, as autonomous robotic solutions become more widely deployed, more attention is put to connectivity and management of larger-scale distributed systems. With increased connectivity, however, also comes an increased risk of cybersecurity threats [7]. A robot featuring different forms or wireless connectivity opens the door to a number of attack vectors. Indeed, recent research has shown that multiple commercial platforms are susceptible to hijacking from different types of interfaces [8]. With robots being cyber-physical systems that often interact with humans and their environment, a security vulnerability becomes both a safety and security risk. In this work, we focus on securing multi-robot interaction and collaborative decision-making from the perspective of a distributed networked system where data is shared and collaborative decisions are made.

The advances in distributed ledger technologies (DLTs), which offer consensus mechanisms among multiple untrusted parties, have made them widely used in interconnected network of devices in recent years. In addition to providing a consensus mechanism, DLT integration also introduces identity management, tamper-proof logging, and smart contract execution, all of which can benefit distributed robotic systems [9]. The majority of solutions for the IoT and robotic applications

Farhad Keramat, Jorge Peña Queralta and Tomi Westerlund are with the Turku Intelligent Embedded and Robotic Systems (TIERS) Lab, University of Turku, Turku, Finland, e-mails: {fakera, jopequ, tovewe}@utu.fi.
Copyright (c) 2023 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3257984

in the literature rely on Ethereum smart contracts [10], [11]. However, there are limitations in terms of throughput and tolerance against network partitioning. Network partitioning is one of the challenges in the integration of DLTs in distributed and mobile robotic systems due to the highly dynamic network topologies and limited bandwidths available. Multiple works in the literature already show the potential of blockchain technology for managing byzantine behaviour or consensus in swarms of robots [12], [13]. In contrast to the majority of Ethereum-based systems, IOTA has already been identified as a solution to address these issues [14], but smart contracts have not been available until very recently.

In this paper, we propose a methodology to achieve collective decision-making in distributed robotic applications in a partition-tolerant manner. We achieve this by utilizing IOTA smart contract platform. This methodology inherently integrates, as well, byzantine-tolerant processes. In addition, we propose a novel architecture to integrate IOTA's two-layer structure with ROS 2.To the best of our knowledge, this is the first approach to integrate IOTA's smart contracts with ROS 2 to leverage DLT in multi-robot systems. We chose a distributed collaborative mapping task to demonstrate how the proposed methodology can be applied on top of our architecture. The distributed collaborative mapping task is simulated in larger-scale in Gazebo as well as with a real-world experiment. The implementations are open-source as the first integration of ROS 2 with IOTA's smart contract.

The core research questions and challenges in current solutions that we address in this paper are twofold. First, how to leverage blockchain technology for real-world mobile networked systems where connectivity might be intermmitent. Architectures based on directed acyclic graphs (DAG) have clear benefits in this direction; however, it is not evident how to achieve resilience against unstable connectivity conditions through network partition tolerance. Additionally, we look into how the partition tolerance brings additional benefits from the perspective of data loss in linear blockchain architectures. Second, we investigate how such a solution can be integrated with ROS 2 for distributed robotic systems, and we design and implement an approach to divide data and logic across the two layers of IOTA to ensure network security. In summary, the main contributions of this work are the following:

(i) to address the challenge of connectivity, we propose, to the best of our knoedge the first general design methodology for DAG-based blockchain and robotic systems. A DAG-based solution does not necessarily imply partition-tolerance. Here, we show how to structure the chains to achieve partition-tolerant collective decision-making, which also inherits byzantine tolerance from the blockchain,
(ii) an architectural design for integrating IOTA and ROS 2 for distributed multi-robot systems, and
(iii) a demonstration of the applicability of the proposed methodology and architecture for a multi-robot collaborative mapping application, where we demonstrate the benefits with respect to existing methods based on Ethereum, addressing the second challenge described above.

The rest of the paper is organized as follows. Section II introduces previous research in the collective decision-making problems and the use of distributed ledger technologies for robotic systems and introduces the key concepts behind smart contracts in both Ethereum and IOTA. Then, in Section IV we describe the methodology to make partition-tolerant decision-making protocols and how to design approaches for integrating IOTA and ROS 2 into the same framework. A partition-tolerant fault-tolerant distributed collaborative mapping algorithm is shown in Section V, with a comparison between Ethereum and IOTA. A discussion on scalability and future potential appears in Section VI. Finally, Section VII concludes the work and lays out the directions for future work.

II. Related Works

This section briefly reviews the literature in robot cybersecurity, and the potential of blockchain and other distributed ledger technologies for securing and building trust in robot swarms.

Multiple research efforts have been directed towards securing robotic systems. For example, in [15], Clark et al. study security threats on robots at the hardware, firmware, and application layer and lists possible attacks from spoofing sensor data to denial of service attacks. In [16], the focus is on the impact of the security attacks and suggests some countermeasures. In another work, Higgins et al. look at both robotics and security perspectives [17]. In this study, the the authors compare swarm robotics use cases to similar technologies in order to find the unique features making similar security measurements unfeasible or ineffective. In general, the security of swarm robotics is very crucial in defence, healthcare, environmental, and commercial applications [18].

Collective decision-making is an essential element in swarm robotics and have studied extensively. In these studies, all robots in the swarm are assumed to be honest and protocol obedient [19]. But a single intruder robot can easily affect the entire system's decision-making. For this reason, recent studies have also taken into account probable faulty robots to make their approach resilient against them. Sargeant and Tomlinson [20] give a generic swarm model and how a malicious intruder can be modeled in this context. In [21], Zikratov et al. propose a dynamic trust management framework that enables robots in an ad-hoc network to detect a compromised device and an access control unit that expects newly joined members to behave honestly up to a certain time to participate in decision-making tasks. For multi-robot systems with time-varying communication graphs dealing with malicious parties is more challenging. In [22], a consensus approach is proposed, resilient if communication graphs reunite in a bounded time period. The method proposed in this work, is a more general approach to solve this kind of issues.

Despite the recent research efforts to design and develop decision-making methods that are fault-tolerant, the literature contains mainly specialized approaches for specific use cases or applications scenarios. To the best of our knowledge, there is a lack of a generic and scalable collaborative decision-making framework for distributed robotic systems. Within this domain, an early work by Castelló Ferrer pointed at the applicability of blockchain technologies in swarm robotics [23].

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3257984

The study discusses how this technology benefits security, distributed decision-making, and new business models in robotics. The work also points out that applying blockchain to resource constraint devices (e.g., mobile robots) can be challenging. However, blockchain technology is still presented as the potential infrastructure to ensure security and safety regulations for robotics. In a more recent work, Afanasyev et al. list open issues in combining blockchain and robotics and certain application scenarios [24]. One of the benefits of blockchain technology is that robots can be assigned specific tasks through smart contracts. In our study, we present smart contracts as the backbone of collaborative decision-making. A proof of concept showcasing blockchain in robotic swarms was demonstrated by Strobel et al. in [12]. In the experiments in [12], Robots in a swarm collaboratively reach an agreement on the most common tile color in an environment with white and black tiles. As opposed to conventional methods, the proposed blockchain-based solution on Ethereum [25] can tolerate malicious behavior. Another example of collective decision-making in swarm robotics in [13] presents an in-depth study of following-the-leader problems. In swarm robotics literature, Ethereum is the most commonly used blockchain platform. However, it comes with limitations in terms of scalability and deployment in embedded systems. Other platforms have emerged that potentially solve some of these issues, such as Hyperledger Fabric [26] and IOTA [27]. These have already been explored in some works, albeit more limited experiments have been demonstrated [28], [14], [29].

From a system design perspective, blockchain technology brings benefits in terms of immutability of past data, and distributed decision making, among others. However, it also brings new challenges. In most existing use cases, blockchains are deployed between nodes (e.g., computers or servers) with a stable network connection. This assumption, nonetheless, does not necessarily hold when operating swarms of mobile robots. To address the issue of potentially intermittent connectivity, Tran et al. proposed SwarmDAG [30]. SwarmDAG is a system-level design based on directed acyclic graphs (DAGs) that incorporates a membership management system to handle new members. Early solutions like SwarmDAG, however, become vulnerable to security issues that traditional blockchains already solve (e.g., Sybil attacks). Next-generation blockchain systems that are intrinsically based on DAGs, such as IOTA, are able to provide both scalability and security. For example, a surveillance system is presented in [14] that also tolerates partitioning within the network while maintaining secure consensus with IOTA. The research in [14] was carried out before smart contracts were developed for IOTA. The IOTA foundation has now introduced IOTA smart contracts that run on chains over the core DAG structure. The technology is therefore now ready for more complex designs and the integration of distributed decision-making processes, taking advantage of smart contracts and asynchronous calls in IOTA. Compared to the state-of-the-art in blockchain-based robotic systems, we propose in this paper a novel approach with a general methodology for making virtually any decision-making problem in a distributed and partition-tolerant manner, given that it can be implemented as a smart contract with a series of constraints. This allows for byzantine-tolerant consensus in multi-robot systems and robot swarms without strong connectivity requirements. At the same time, this solution achieves higher degrees of scalability when compared to traditional blockchains that form the vast majority of the work to date in DLTs within the robotics field.

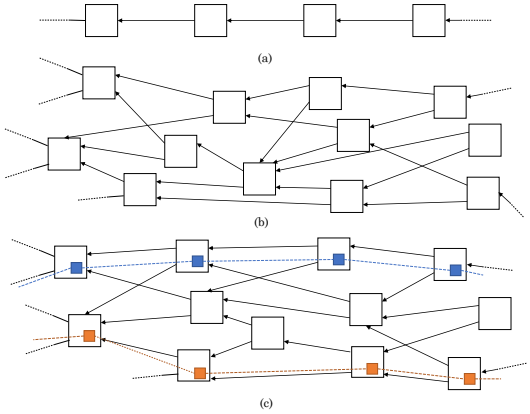


Fig. 2: Illustration of (a) a traditional blockchain, (b) the IOTA Tangle, and (c) the two-layer ISCP architecture where we highlight two possible chains anchored to different Table nodes.

III. Background

Blockchain systems, a subset of the wider domain of DLTs, have grown in popularity over the past few years, partly owing to the public interest around cryptocurrencies. Bitcoin [31] was the first cryptocurrency that removed the trust on a third party to conduct a transaction between two entities. Although financial transactions were the initial objective, blockchains can be used in a variety of use cases. In addition to enabling distributed decision-making, as we discussed earlier, blockchains also serve as a tamper-proof log. Blockchain technology has been used in some robotics research just for logging purposes to have a reliable history of events [32]. Smart contracts, which are computer programs automatically executed when a set of conditions is fulfilled, have opened up new possibilities for distributed applications (DApps). In the following, we describe the key concepts behind the more traditional Ethereum blockchain and the DAG-based IOTA architecture. These different design approaches are illustrated in Fig. 2.

Ethereum: Ethereum made a significant impact on distributed ledger technologies by introducing Turing-complete smart contracts. Solidity is Ethereum's programming language exclusively designed for coding smart contracts. A Turing-complete smart contract has enabled Ethereum to be widely adopted in a wide range of domains. Researchers have also exploited Ethereum's capabilities within multi-robot systems [12]. While Ethereum introduced new possibilities to blockchain systems, it still uses the classical single-chain structure. Therefore, the intrinsic scalability problem remains. In addition to scalability issues, and considering the perspective of the design of multi-robot systems, tolerating network partitioning is imperative for real-world deployments in many

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3257984

application scenarios. The Ethereum foundation is working on Ethereum 2.0, which is posed to resolve many of the current scalability issues by leveraging *sharding* (a new approach to achieving consensus within subsets of the global network); however, the solutions are not matured yet.

Tangle (IOTA): The Tangle, a DAG-based DLT, was introduced by S. Popov. to solve some of the underlying deficiencies in classic blockchain systems [27]. The Tangle is the underlying structure used by the IOTA DLT. Most of the blockchain systems to date use blocks encapsulating a set of transactions as their primary data structure. These data blocks are then usually connected as a linked list by using the hash of each block as the linking element between consecutive entries in the list. In the Tangle, transactions themselves are the primary data structures. Using individual transactions as the primary data structure enables even nodes with limited resources to participate in the consensus. The DAG structure is then generated as each transaction must refer to two previous unconfirmed transactions based on the view of the Tangle that the node that issues it has. The core idea behind the Tangle is that keeping the ledger on a graph rather than a single (linear) chain would allow for a certain level of flexibility in terms of network partitioning. This makes IOTA, a priori, an excellent DLT solution for multi-robot systems [30], [14].

Shimmer (IOTA 2.0): The first version of IOTA focused on making the graph-based data structure functional while preserving the security standards of most DLT solutions. In this version, only basic transactions were possible (e.g., financial transactions, such as a exchange of tokens, or data publishing for IoT devices). To keep the Tangle stable and secure, a centralized Coordinator managed by the IOTA Foundation was in charge of confirming valid transactions. To achieve full decentralization, the IOTA Foundation redesigned the Tangle and launched Shimmer as the second version of IOTA. GoShimmer is the Go implementation of Shimmer clients, which will be used in this work.

Wasp (IOTA Smart Contracts): Due to the graph-based data structure in the Tangle, embedding a smart contract mechanism in IOTA was a bigger challenge than in traditional blockchain. In general terms, smart contracts are made possible in a blockchain through a state machine that has a state that can be altered by entering a new block. Such state machine requires a global state and is therefore not directly *embeddable* within the Tangle. To solve this issue, the IOTA Foundation uses the Tangle as a first layer, on top of which they introduce the IOTA Smart Contract Platform (ISCP) as a second layer. Wasp refers to the Go implementation of the ISCP client. ISCP clients or Wasp nodes can create a chain in this second layer. Each of these chains can be compared to an Ethereum blockchain, in this case having every block *anchored* to the first layer. Other Wasp nodes can join the chain, and all the Wasp nodes participating in a chain form a *chain committee*. Similar to an Ethereum blockchain, committee members can run smart contracts in the chains they belong to. Each *chain committee* has a finite number of members, meaning they can run a byzantine fault tolerant (BFT) algorithm to reach consensus. BFT consensus algorithms can tolerate at most one-third of byzantine members. It is possible to make virtually any number of chains on top of the first layer with different committees. In addition, these chains can interact with each other, referred to as asynchronous calls in ISCP. Asynchronous calls enable smart contracts to call a method of another smart contract in a different chain. It is worth mentioning that, while global connectivity is not required at all times within the Tangle, running a smart contract requires consensus to be reached by members of the corresponding Wasp committee. Therefore, at least two-thirds of the nodes participating in a chain need to be in a common network partition when chain blocks are committed. This requirement does not extend to two chains interacting through asynchronous calls except when the calls occur.

Robotics Middleware: the Robot Operating System (ROS) is the de-facto standard in today's autonomous robots [33]. From the perspective of multi-robot systems and distributed networked systems, the original ROS 1 version has certain limitations, mainly due to the existence of a certain node managing the interaction between the different actors in the system. The new ROS 2 version solves this with the introduction of the data distribution service (DDS) standard for the lower-level communication middleware. DDS also provides a security extension empowering ROS 2 itself. Only ROS 2 is a natural selection for integration with distributed ledger technologies. In addition to the DLT platform in use, the distributed communication and security that DDS enable are crucial features for a secure and trustable system. This work does not aim at replacing those features, but instead complimenting them with an additional channel for building trust and implementing collaborative decision-making processes. ROS 2 already provides tools for data encryption and data access control.

In summary, to address the network partitioning problem in multi-robot systems, and enable dynamic network topologies, IOTA is a promising solution. This has already been showcased with a proof of concept in the literature [14], however, with the lack of smart contracts for more complex integrations. With an IOTA-based system, disconnected robots or separate groups of robots can still operate on the same global Tangle, with the corresponding transactions in separate subgraphs that are merged whenever global connectivity is regained. With the introduction of ISCP and the asynchronous calls, in addition to this, we can also design systems that run distributed applications inherently able to tolerate network partitioning. In this paper, we propose a general approach for making distributed, partition-tolerant decision-making tasks through IOTA smart contracts. We design such approach to be integrated with ROS 2 in a seamless way. This integration is two-directional, with ROS 2 feeding data to the IOTA Tangle and IOTA smart contracts being used to implement functionality that replaces distributed ROS 2 nodes. In particular, we exploit the asynchronous calls for *connecting* chains that *live* in different network partitions.

IV. Methodology

This section covers the overall system design, including how robots operate with the two-layered IOTA architecture and how IOTA is bridged with ROS 2. The section focuses

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3257984

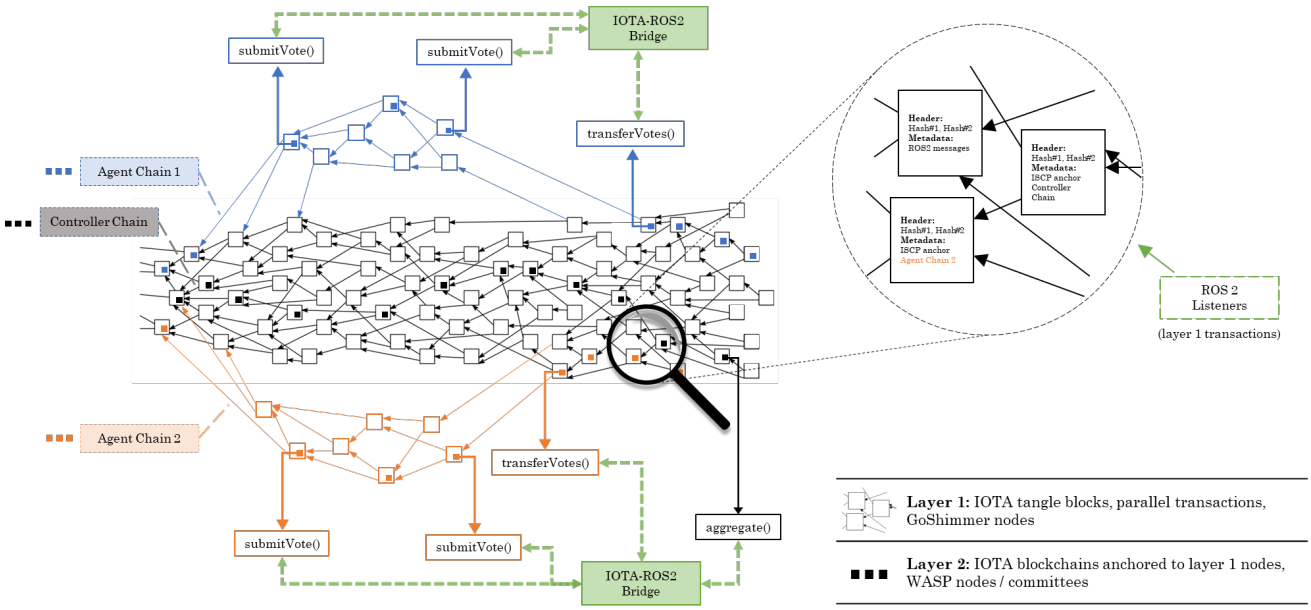


Fig. 3: Illustration of the integration of ROS 2 with the IOTA tangle (layer 1) and IOTA smart contracts (layer 2) enabling secure, distributed and partition-tolerant decision-making. The main network partition, represented in black, includes the core CONTROLLER SWARM chain as well as other nodes with global connectivity. In orange and blue we illustrate two examples of transactions (both layer-1 and layer-2 chains) associated with mobile robots with intermittent connectivity and thus create their own network partitions. These partitions can also be formed by groups of robots that remain locally connected, but globally disconnected from the rest of the network. This figure does not include GoShimmer and Wasp nodes to prevent further complexity. Both layer-1 and layer-2 nodes are bridged to ROS 2, with layer-1 nodes being passive listeners that dump ROS 2 data to the tangle. Layer-2 nodes have full bi-directional communication between IOTA and ROS 2 and implement the integration interface.

on how smart contracts are designed and deployed to allow for network partitions and intermittent connectivity while maintaining global consensus at the end of a distributed multi-robot mission.

A. Problem Definition

Collaborative decision-making processes in multi robot systems often refer to any method that requires the combination of input, or data, from different robots and processes the data or makes a decision based on it in a decentralized manner. Examples include distributed role allocation algorithms, distributed perception, cooperative mapping, or decentralized formation control, among others. We assume in all cases that robots operate as individual entities and that the decision-making process is not governed by a central authority. These processes are used to achieve agreement and specialization in multi-robot systems [5]. In this work, we focus only on reaching agreements on a series or variables, often referred to as consensus problems.

The core objective of this work is to provide a framework for collaborative decision-making in a multi-robot system based on IOTA. To this end, we integrate IOTA smart contracts with ROS 2 nodes, in a way that data fed to IOTA from ROS 2 is processed within smart contracts. The results of such process are then fed back to ROS 2 topics, even though ROS services or actions could also be implemented in the future. To achieve this goal, this section covers the first two contributions of our work:

i. First, we introduce a methodology and architecture for integrating IOTA with ROS 2 nodes. We explain how ROS 2 data and processes can be either be part of the IOTA Tangle or be implemented as smart contracts in IOTA's second layer.
ii. Second, we propose an strategy for enabling partition-tolerant decision-making through the combination of multiple smart contracts in different chains, each existing in its own network partition within the Tangle.

In addition to introducing this approach, our experimental results reported in the next section focus on a proof of concept of the proposed methodology with a cooperative mapping use case, covering the third contribution of our work.

B. System Architecture: Integrating IOTA and ROS 2

To leverage IOTA's smart contracts in the multi-robot systems, we propose an integration of ROS 2 with the two-layer IOTA architecture illustrated in Fig. 3 as described in the following:

GoShimmer network: Designed for lightweight IoT nodes, every robot in the system can run a shimmer node. A cloud fleet management system, or fixed sensors and infrastructure with connectivity can also be part of the GoShimmer network in IOTA's layer 1. Because of how IOTA is designed, the more nodes that participate in the network, the more secure that IOTA's Tangle is. Within the system performance limits, the GoShimmer network is therefore a good place to publish higher frequency transactions. For example, raw sensor data from the robots can be published in the layer 1. This data can also be used for validation in different processes later on. In Fig. 3, the GoShimmer network is not visible but is represented by the layer 1 transactions.

GoShimmer nodes running in the robots are thus also ROS 2 *listeners*, i.e., they subscribe to a topic and re-publish the data

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3257984

in the Tangle. The nodes can also be configured to publish data from the Tangle (e.g., metadata from other transactions, data about the Tangle topology, or even the node's own real-time performance data). To implement this, we integrate the GoShimmer library with *rclgo*, the ROS 2 client library for Go. Publishing ROS 2 messages on the Tangle would generate a tamper-proof log of events.

Wasp committees: Running a Wasp node on every robot in the system is unnecessary; instead, we propose to run Wasp nodes on the robots with higher processing power or the cloud fleet management system. In order to maintain security, Wasp nodes must be distributed evenly throughout the system. We propose that at least $3f + 1$ robots with higher processing power run a Wasp node in each subgroup of robots that might have intermittent connectivity to tolerate f malicious robot. Every Wasp node should be connected to a GoShimmer node to interact with L1. Since we proposed to run a GoShimmer node on every robot, a Wasp node should connect to the GoShimmer node running on the same robot.

According to applications running on the multi-robot system, Wasp nodes should create chains and form committees. All the robots working on an application can operate on a single chain in case they don't use interchain asynchronous calls. If the application utilizes interchain asynchronous calls, Wasp nodes should be distributed on the different chains accordingly. Fig. 3 does not illustrate the Wasp committee member nodes, but instead every colored layer 2 chain represents a Wasp committee chain.

ROS 2 bridge: We have implemented a ROS 2 listener node with *rclgo* that subscribes to a topic and forwards these messages or hash of the messages to GoShimmer nodes. GoShimmer nodes will use these data to create new transactions in order to keep the network alive. Keeping these messages on the ledger will provide a complete log of events in multi-robot systems. In addition to a ROS 2 listener node, a ROS 2 talker node can be implemented to read on-ledger transactions and publish them for log reviewing and validating processes.

For Wasp nodes, a similar approach is utilized to interact with ROS 2. A Go script is utilized to call methods of smart contracts and also retrieve the results from it. This script consists of a ROS 2 subscriber which listens on a topic to receive the command to call a method from a smart contract on the chain specified with *ChainID*. If the method has a returning value, the script will retrieve the value and publish it on a predefined topic. Also, ROS 2 actions could be a better alternative instead of using a listener and talker topics.

Figure. 3 illustrates part of the Tangle and the Wasp chain blocks anchored to the Tangle. In the figure, time increases from left to right. Empty squares show L1 transactions created by GoShimmer nodes. In each transaction, the header includes the hash of two referred transactions and a metadata part. This metadata can be dumped ROS 2 messages or the anchored state of a chain. Filled squares inside an empty square shows a block of the L2 chain created by the Wasp committee. Three chains are shown in this figure, colored blue, orange, and black, with only one smart contract deployed on each chain. Methods called from each smart contract are depicted in squares connected by arrows to the block where this call happened. For example, the $submitVote$ method from the orange chain is called twice, and later $transferVotes$ method is called. These method calls are also through the IOTA-ROS2 bridge.

```
pragma solidity ^0.8.7;

contract MainContract {

    VOTE[] votes;

    function submitVote (VOTE memory vote) {
        // Some code goes here
        votes.push(vote);
        // Rest of the code goes here
    }

    function aggregate () returns result {
        // Implementation goes here
        return result;
    }
}
```

Listing 1: General smart contract definition (Ethereum's Solidity) for a collaborative decision making implementation.

C. Partition-Tolerant Collaborative Decision Making

As we have described earlier, current solutions that leverage DLTs in robotics are not able to deal with network partitions. Indeed, if two robots are meant to share data regularly on a blockchain, but they remain disconnected for part of the data gathering process, a problem raises in terms of how can the individual chains be merged. We discuss here how asynchronous calls in the ISCP can solve this issue. Our objective is to enable multiple robots to achieve consensus in terms of global mission parameters or variables (e.g., a shared map or a bijective role allocation), while maintaining local consensus within their internal systems or local subnetwork.

Collaborative decision-making processes can be typically implemented on a smart contract following the example in Listing. 1. This example, for the Ethereum blockchain, has been followed by different works in the literature. The collaborative decision-making process then proceeds as follows. Every entity that is participating in the task has a vote. These votes could be any type of data based on the goal they are going to achieve. Every participant can submit their vote with the $submitVote$ function of the smart contract. $aggregate$ is a view function (only reads data from the ledger and doesn't change anything) used for aggregating the votes to get the final result of consensus.

Such collaborative decision making processes which can be formulated as a smart contract similar to the general form mentioned above can become partition tolerant. Partition tolerance is achieved by the multiple chains introduced by ISCP. In order to make a smart contract partition tolerance, we propose that it should be divided into two separate smart contracts. First, a CONTROLLER SWARM smart contract should be deployed on the CONTROLLER chain. Second, every group of robots that may at some point be disconnected from the core network for a certain amount of time should create their own chain and deploy a new instance of an AGENT SWARM smart contract on the chain.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3257984


```
name: ControllerSwarm
structs:
  VOTE:
    # Data structure of votes goes here
state:
  votes: VOTE[]
  # Other state variables goes here
funcs:
  submitVotes:
    params:
      votes: VOTE[]
  aggregate:
    results:
      result: RESULT
  # Other functions may be also defined here
```

Listing 2: ControllerSwarm smart contract schema.

```
name: AgentSwarm
structs:
  VOTE:
    # Data structure of votes goes here
state:
  votes: VOTE[]
  # Other state variables goes here
funcs:
  submitVote:
    params:
      vote: VOTE
  transferVotes: {}
  # Other functions may also be defined here
```

Listing 3: AGENT SWARM smart contract schema.

The AGENT SWARM contract is responsible to store the votes on the ledger, so that they are saved in case that these robots disconnect from the rest of the network through the $submitVote$ function. In addition, this smart contract provides the $transferVotes$ function that can be called by any member on this chain to transfer the stored votes to the CONTROLLER chain. This function uses an asynchronous call to the $submitVotes$ function of CONTROLLER SWARM smart contract. The CONTROLLER SWARM contract implements the $aggregator$ function that aggregates all the votes from the different chains. These smart contracts are illustrated in Listing. 2 and Listing. 3.

D. Eventual Consistency

Based on Brewer's CAP conjecture for a distributed system [34], we cannot achieve consistency, availability, and partition tolerance simultaneously. As It is proposed to make the collaborative decision making tasks partition tolerant in a distributed manner, the consistency will not be acquired anymore. As mentioned in the previous section, the $submitVotes$ function of the AGENT SWARM smart contract is called when the chain members are again connected to the rest of the network. Therefore, in this case, the system has eventual consistency. Every group of swarms on a chain can be disconnected for a while, but when they are connected to the CONTROLLER chain, the system will reach an eventual consistency. This assumption does not lead to any degradation of the functionality of the robotic system. Indeed, if a robot or group of robots remains disconnected from the CONTROLLER chain, then they can maintain operation within the ledgers defined in their own network partition, unrelated to the rest of the system. If the disconnection is a result of a malfunction, then it is out of the scope of this work to provide connectivity maintenance or recovery methods, with the robotics literature containing solutions to such challenges.

To achieve more robust multi-robot systems using DLTs, it is worthy to prefer partition tolerance over consistency. However it is possible to have eventual consistency that is sufficient in many scenarios. If we want to choose consistency, then we might loose some part of the data. In next section, we will demonstrate a scenario that loosing data how can affect the overall decision making problem.

E. Byzantine Tolerance

The proposed architecture inherits byzantine tolerance from the IOTA smart contract platform. Members of Wasp committees run an instance of a BFT consensus algorithm. The BFT consensus algorithm implies that at most one third of the committee members can be malicious [35]. A Wasp committee's consensus procedure can be halted if byzantine agents make up a third of the members. Nevertheless, two thirds of the members must constitute the majority in order for incorrect data to be entered in the ledger. To preserve the byzantine tolerance of the proposed architecture, every Wasp committee should have at least $3f + 1$ members in order to tolerate f byzantine agents. f is the predetermined security parameter that depends on the required security level.

V. EXPERIMENTAL RESULTS

In this section, we focus on a specific proof of concept to demonstrate the usability and applicability of the proposed methods. The vast majority of indoor mobile robots today use 2D lidars for navigation and mapping, with mapping being a key step for a robot to achieve situational awareness. We choose cooperative mapping as a representative task involving consensus (agreement on the final map of the operational environment) and the aggregation of data from different robots (the individual maps). This task includes role allocation, and agreement between the entities. At the same time, the individual maps can be checked against the real-time sensor data published in the Tangle. Finally, collaborative mapping is an excellent example of a consensus problem requiring a partition-toleran implementation for real-world applications. The network partitioning is highly probable due to undiscovered heterogeneous environments. Also, byzantine entities might alter the collaboratively created map to exclude some areas to be navigated, or even expose other robots to real hazards by altering their local maps. In summary, we choose a simple and intuitive approach for collaborative mapping to demonstrate the method proposed in the previous section.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3257984

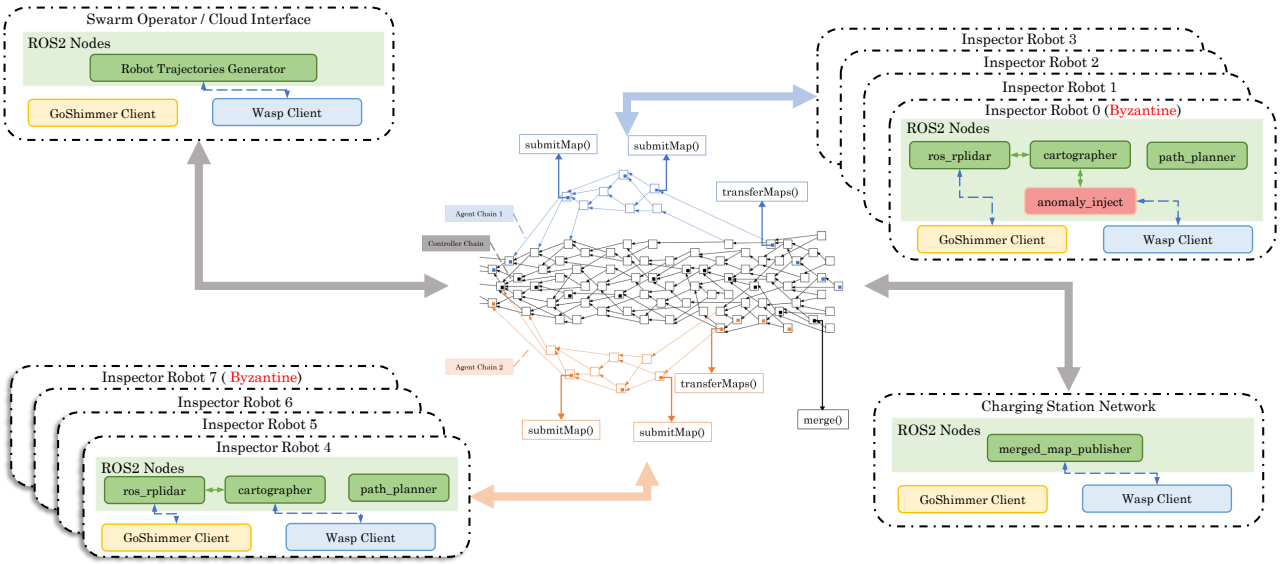


Fig. 4: Integration of ROS 2 and IOTA smart contract for the cooperative mapping application use case. In this figure, every box represents a robot or other nodes participating in the network. Every robot runs a GoShimmer client, Wasp client, and some ROS 2 nodes. The other nodes also runs their own GoShimmer and Wasp clients. The bold colored arrows shows the interaction of the clients with the ledger.

Algorithm 1: Compare two maps

Input: $M_{l\times w}, M'_{l\times w}$
Output: $True$ or $False$
$comply \leftarrow True$;
$T \leftarrow 0.2$;
$T' \leftarrow 0.2$;
foreach $W_{n\times n}$ ***in*** $M_{l\times w}$ **do**
  foreach $W'_{n\times n}$ ***in*** $M'_{l\times w}$ **do**
    $\mathbb{S} \leftarrow \{w_{i\times j} = w'_{i\times j}, w_{i\times j} \neq U\}$;
    $\mathbb{D} \leftarrow \{w_{i\times j} \neq w'_{i\times j}, w_{i\times j} \neq U, w'_{i\times j} \neq U\}$;
    $\mathbb{U} \leftarrow \{w_{i\times j} = U\}$;
    $\mathbb{U}' \leftarrow \{w'_{i\times j} = U\}$;
    if $\frac{|\mathbb{U}|}{n^2} < T \wedge \frac{|\mathbb{U}'|}{n^2} < T \wedge \frac{|\mathbb{D}|}{|\mathbb{D}|+|\mathbb{S}|} > T'$ **then**
      $comply \leftarrow False$
return *comply*

Algorithm 2: Smart contract map submission

Input: $M_{l\times w}$
Data: $\mathbb{M}$ List of submitted maps
$complies \leftarrow 0$;
$contracts \leftarrow 0$;
foreach $M'_{l\times w}$ ***in*** $\mathbb{M}$ **do**
  $comply \leftarrow compare\ (M_{l\times w}, M'_{l\times w})$;
  if *comply* **then**
    $complies \leftarrow complies + 1$;
    $complied(M'_{l\times w}) \leftarrow complied(M'_{l\times w}) + 1$;
  else
    $contracts \leftarrow contracts + 1$;
    $contracted(M'_{l\times w}) \leftarrow contracted(M'_{l\times w}) + 1$;
$complied(M_{l\times w}) \leftarrow complies$;
$contracted(M_{l\times w}) \leftarrow contracts$;
$\mathbb{M}.append(M_{l\times w})$;

A. Partition-Tolerant Collaborative Mapping

For the collaborative mapping application, we consider a rectangular area to be mapped. We assume that the rectangle has a known size of $L \times W$. We also assume that there are multiple robots that are going to participate in this task as a service. The area is divided into $l \times w$ sized cells. The cells are considered to be small enough so that the topology does not change significantly from different viewpoints, and so that a single lidar scan is enough for mapping the cell. Each cell should be visited by $k \geq 3f + 1$ robots, such that at most f out of them are byzantine.

First, robots should register their identity to take part in the mapping task. Second, each robot is assigned to a randomly selected cell in the area. Third, the robot travels to the assigned cell to map it. Fourth, the robot submits the built map to the smart contract. The smart contract based on the received map merges them.

We define every local map as a 2D matrix of size $M_{l\times w}$. The resolution for creating the map is a fixed and predefined value. Every entry of the matrix is filled by the robot with out of three values ($\{O, F, U\}$) representing occupied, free, and unknown, respectively.

To implement this partition-tolerant collaborative mapping on a smart contract, we define five main functions. First, a role allocation function to assign a random cell to each robot which is going to participate in the inspection. This function is responsible to assign each cell for at least $k \geq 3f + 1$ robots to tolerate f byzantine robot. Second, a function that is

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3257984

Algorithm 3: Smart contract merge function

Data: $\mathbb{M}$ List of submitted maps
$global\_map \leftarrow \emptyset$;
foreach $M_{l\times w}$ ***in*** $\mathbb{M}$ **do**
  if *complied*$(M'_{l\times w}) <$ *contracted*$(M_{l\times w})$ **then**
    agent$(M_{l\times w}) \leftarrow byzantine$;
  if *agent*$(M_{l\times w})$ ***is not*** $byzantine$ **then**
    $global\_map$.*append*$(M_{l\times w})$;
return $global\_map$;

responsible of comparing two maps as defined in Algorithm 1. This function will return a true or false value indicating where the two input maps are complying (true) or conflicting (false). This comparison function will then be used in the submission function. The third function is the submission function, defined in Algorithm 2. For every new map that is submitted to the smart contract, it is compared to the all previous submitted ones. For each map, two values are stored. The $comply()$ value of the map, representing the number of maps complying with this map, and the $conflict()$, defined in the same way for the number of maps with which it conflicts.

The fifth function, namely map merging, is defined in Algorithm 3. The merging function is responsible for merging the maps. In this context, merging is the $aggregate$ function of the general smart contract defined in the previous section. Equivalently, the generic $votes$ defined earlier are now maps in this application, and the generic $submitVote$ function is implemented through the map submission function.

B. Multi-Robot Gazebo Simulation

For the purpose of testing the proposed partition-tolerant collaborative mapping task with a larger number of robots, a small town-like environment is designed in the Gazebo simulator. The simulation environment is shown in Fig. 5. Eight robots are deployed in the simulator to map the town. We set two of these robots as byzantine agents. Their objective is to try to *fool* other robots by inserting certain random walls on their local town maps. Robots navigate the town on predefined paths while using a 2D lidar to scan the surrounding environment. Based on the output of the 2D lidar and the odometry data generated by Gazebo simulator, the Cartographer ROS/2,2 implementation is used for mapping [36]. A ROS 2 node is implemented in order to emulate the behavior of byzantine agents. This node is deployed on predefined byzantine agents and subscribes to the output map of Cartographer, adds random walls to the map, and then republishes it. The final local maps (either directly the output of the Cartographer node or the altered maps generated by the nodes running in the byzantine agents) are submitted by robots to the Controller Swarm smart contract. The maps that are submitted by each robot are shown in Fig. 5, each covering a different area of the town and with the lack of a global view. After inspecting the assigned cell, the robot will submit its map to the smart contract. Another ROS 2 node is implemented to call the merge function of the smart contract and publish the results on a ROS 2 topic. The merged map in different stages is illustrated in Fig. 6. In this experiment *Robot 0* and *Robot 7* are byzantine robots. The maps submitted by these robots are eliminated in the overall merged map when enough data is available about the sections where byzantine agents alter the real maps. It is worth noting at this point that inserted data may temporarily appear in the merged map until a high enough number of robots submit a map that conflicts with the map submitted by the byzantine robot. For a specific application and collaborative decision-making process implementation, a byzantine behaviour can be potentially designed in an adversarial manner to surpass the detection mechanisms in the smart contract. However, our focus here is not on defining a robust byzantine agent detection strategy but instead on introducing a general way of building trust with IOTA through a partition-tolerant implementation. Therefore, more advanced smart contracts specifically designed to detect altered data are out of the scope of this work.

The simulation results show that our method effectively enables both byzantine-tolerant and partition-tolerant collaborative mapping under certain conditions. Through the simulation, we effectively emulate network disconnections as robots only submit their local maps once their path is finalized. In this implementation, each robot individually forms its own Wasp committee. In practice, several robots operating in nearby cells can form a common Wasp committee.

C. Real-World Experiment

For real-world experiments we used a Dashgo indoor UGV with an RPLIDAR A1 installed on top of it. For localization, we rely on an Optitrack MOCAP system providing global, real-time, 6D pose estimation through ROS 2 topics. We create a small maze to be navigated by robots. For the sake of simplicity, we map different areas of the maze iteratively, adding or removing physical elements in between to emulate altered sensor data. The maze and the Dashgo UGV assembly can be seen in Fig. 8. In practice, the same robot is operated four times to create the data for four different virtual agents. This is done without loss of generality from the perspective of the methods introduced in this paper. Four different GoShimmer and Wasp nodes are deployed in two sub-networks to demonstrate the partition tolerance of the methods, with these sub-networks simulating network disconnections. These nodes are deployed on an UP Squared board placed on top of the Dashgo UGV. Also, an UP Xtreme board is used to play the role of charging station with its own GoShimmer and Wasp nodes. The hardware specifications of these boards are listed in Table. I. In this experiment, *Robot 3* will simulate the network disconnection. GoShimmer and Wasp nodes associated to *Robot 3* are deployed on a Docker network. The *Robot 3* Wasp node will create a chain and deploy the AGENT SWARM smart contract. Other GoShimmer and Wasp nodes will use another Docker network with a chain which another instance of AGENT SWARM smart contract is deployed on. GoShimmer and Wasp nodes of the charging station also has their own Docker network and CONTROLLER SWARM smart contract deployed on a chain. These Docker networks communicate through an Docker overlay network. The White box in Fig. 8 is inserted as an anomaly while *Robot 1* is navigating, which corresponds to the byzantine robot mapping the area.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3257984

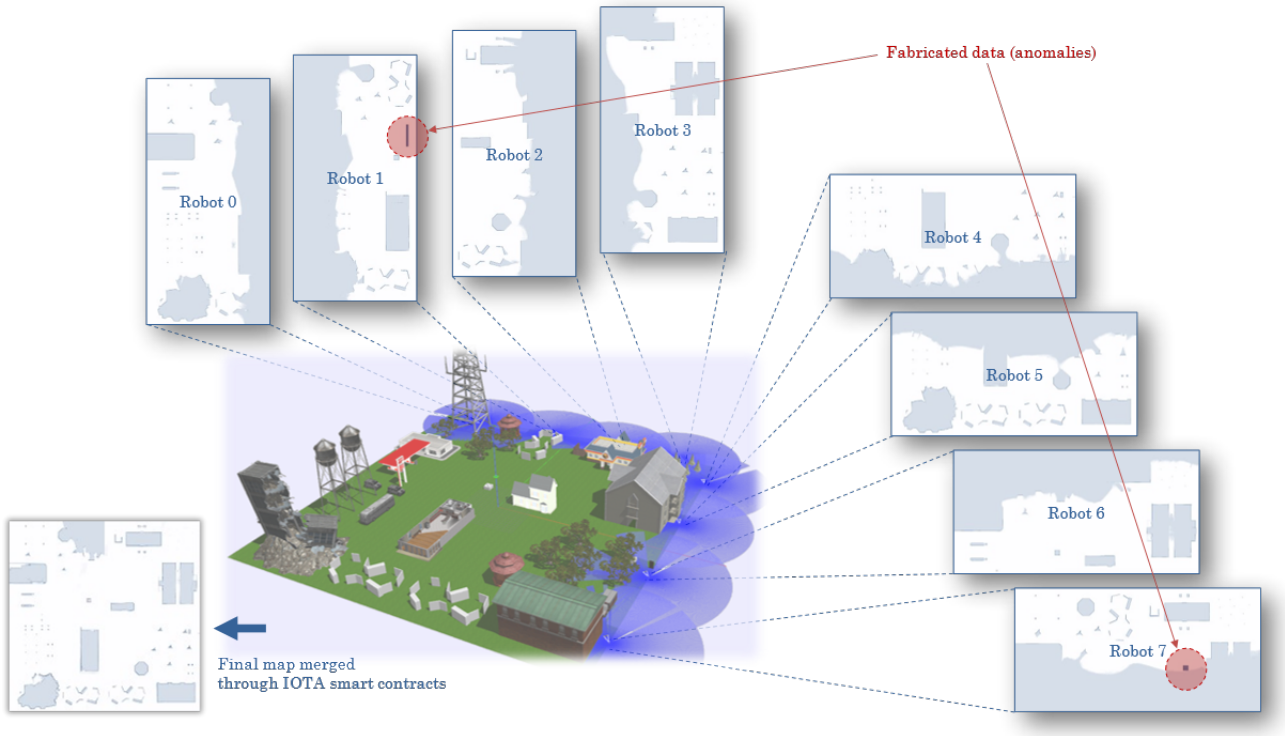


Fig. 5: Mapping results in the Gazebo simulation where eight robots map a small town. The dark blue lines represent the lidar scan range for each of the robots; the robot paths are omitted to improve visualization. Each individual map is generated by the robots locally using cartographer. The final map obtained with IOTA smart contracts is shown in the bottom left, where the anomalies detected in the data submitted by two of the robots have been effectively eliminated.

TABLE I: Hardware Specifications of UP Squared and Xtreme boards used for real-world experiments.

	UP Squared	**Up Xtreme**
CPU	Intel Atom x7-E3950	Intel Core i7-8665UE
Mem / Disk	8 GB / 64 GB	16 GB / 64 GB

TABLE II: CPU and memory consumption of GoShimmer nodes on UP Squared and Xtreme boards

	UP Squared	**Up Xtreme**
GoShimmer avg. CPU	25.85%	20.16%
GoShimmer avg. MEM	213.65 MB	228.8 MB

The raw lidar scan is published by GoShimmer nodes on the Tangle. *ROS 2 LaserScan* messages, which are approximately 1.5 KB each, are published with a 5.5 Hz rate. The CPU and memory consumption of every board is reported in Table. II. The map created by each robot is illustrated in Fig. 9.

To see the difference between IOTA and Ethereum and how partition-tolerance can benefit this mapping process, the maps are submitted to smart contracts deployed on both Ethereum and IOTA networks. The map merging result of the Ethereum and IOTA smart contracts are illustrated in Fig. 9. In the merged map generated through the Ethereum smart contract we can observe that the map submitted by the robot which was in different sub-network was eliminated. This is effectively caused because only one of the two chains created when robots are disconnected remains at the end of the mission (Ethereum discards the shortest chain, also defined as the chain with the lowest accumulated computational complexity). With the IOTA-based implementation, this part of the map is included and the anomaly wall effectively removed at the same time.

These results demonstrate the following. First, the IOTA-based implementation is superior to more traditional Ethereum implementation in terms of supporting network partitions. In practice, this means that whenever robots disconnect in the real world, their data is not necessarily lost. We quantify this with the percentage of mapped area that results from IOTA and Ethereum smart contracts and listed in Table III. IOTA smart contracts achieve higher percentage of mapped area than Ethereum smart contracts and even the union of robot maps by masking the anomalies.

With the vast majority of the literature in DLT integrations for robotics relying on traditional blockchains, this work solves one of the key practical problems stopping more widespread use of this technology. Second, the IOTA-based implementation is effectively able to detect and neutralize the behaviour from the byzantine agents. This is a novel results from the point of view of the technology in use, albeit several works in the literature showcase such ability in Ethereum-based solutions. Nonetheless, this result also shows that the IOTA-based methods maintain the functionality of existing research while extending applicability. Finally, a third conclusion from the reported results is that this work opens the door to more scalable and wider use of DLTs within distributed robotic systems. While this is not proved in this paper, the DAG-

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3257984

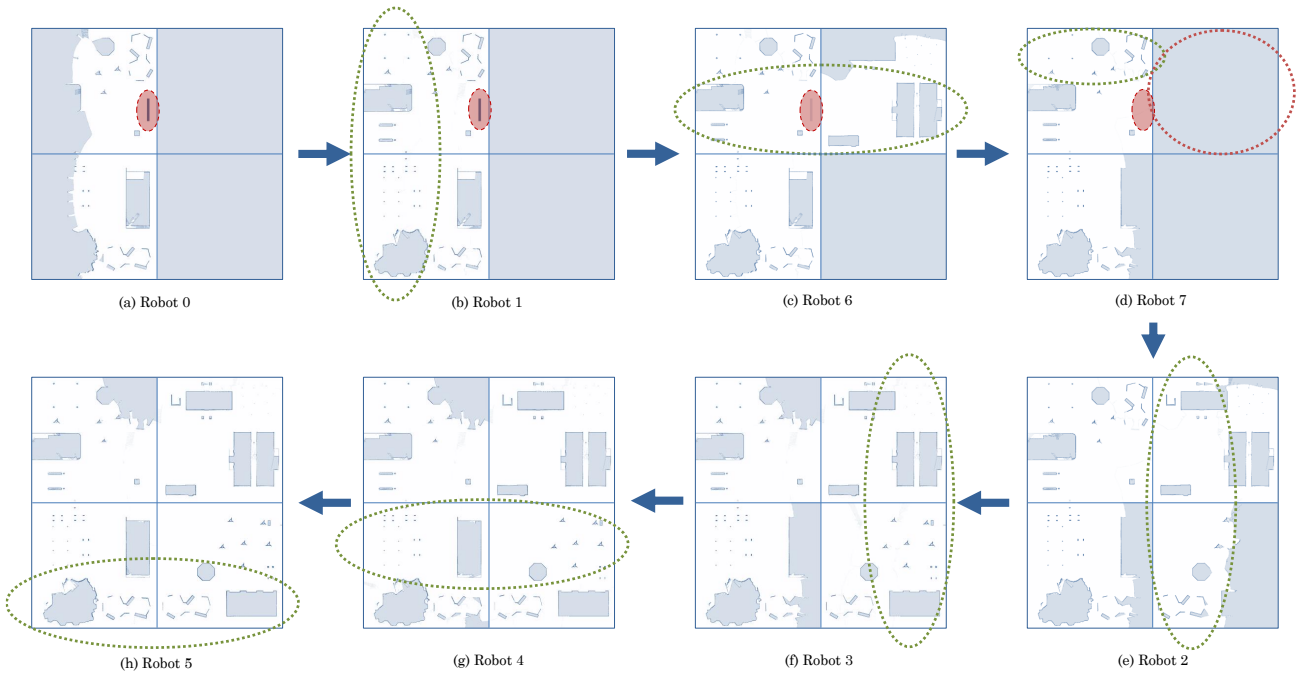


Fig. 6: Illustration of the map merging process as different robots submit their local maps in time. Once *Robot 0* and *Robot 1* have submitted their maps to the smart contract, the overall map is updated as these two maps are not conflicting. The process continues with other local maps, with anomalies being effectively eliminated once enough data is available about the corresponding map cells. An alternative implementation might wait for a certain number of robots to submit their maps before merging the data, without changing the core processes.

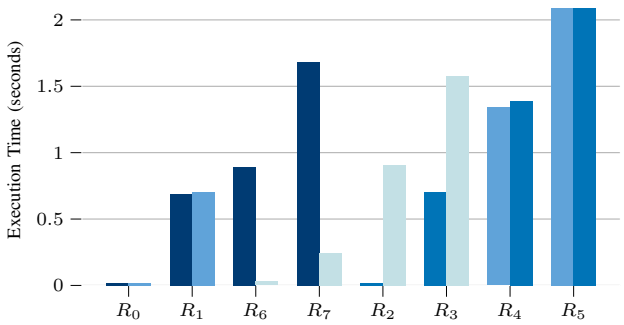


Fig. 7: Execution time of $SubmitVotes$ method of CONTROLLER SWARM smart contract for each robot. The simulation environment is divided in four cells or quadrants. Each robot navigates mainly through two of the quadrants of the map and submits two separate maps. The four bar colors in the graph represent these four cells. Every map submitted is compared on a cell-by-cell basis only when there is an intersection between maps.

Fig. 8: Portion of the maze created for the mapping experiments. The Dashgo UGV is also seen in the image.

based architecture of IOTA and the nature of its solutions with respect to Ethereum and other traditional blockchains mean that more scalable solutions for more resource-constrained real-world devices can be designed and developed. It is worth noting that the Ethereum community is also working, with different methods, towards a more scalable DLT, and proof-of-authority implementations already bring some benefits, albeit not the network partition tolerance we look for in this work.

D. Scalability Experiment

In addition to the experiments showcasing functionality and properties of the system, in order to observe the scalability of the system we repeated a similar experiment to Section V-B, now with a varying numbers of nodes. In these experiments, 4, 8, 12, and 16 robots execute GoShimmer nodes while the lidar scan data of each robot is published on the ledger for one hour. To compare the size of the raw data being published over the ledger with the resources used by the GoShimmer node, we recorded a *ROS bag* which contains all the published messages over the ROS 2 middleware. The memory, hard disk, and bandwidth consumption of the nodes are measured with the Docker *stats* tool after one hour of operation. In Fig. 10, we show the average of these values for all nodes. The memory consumption remains almost constant for larger number of nodes. Even though hard disk usage and bandwidth increase

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3257984

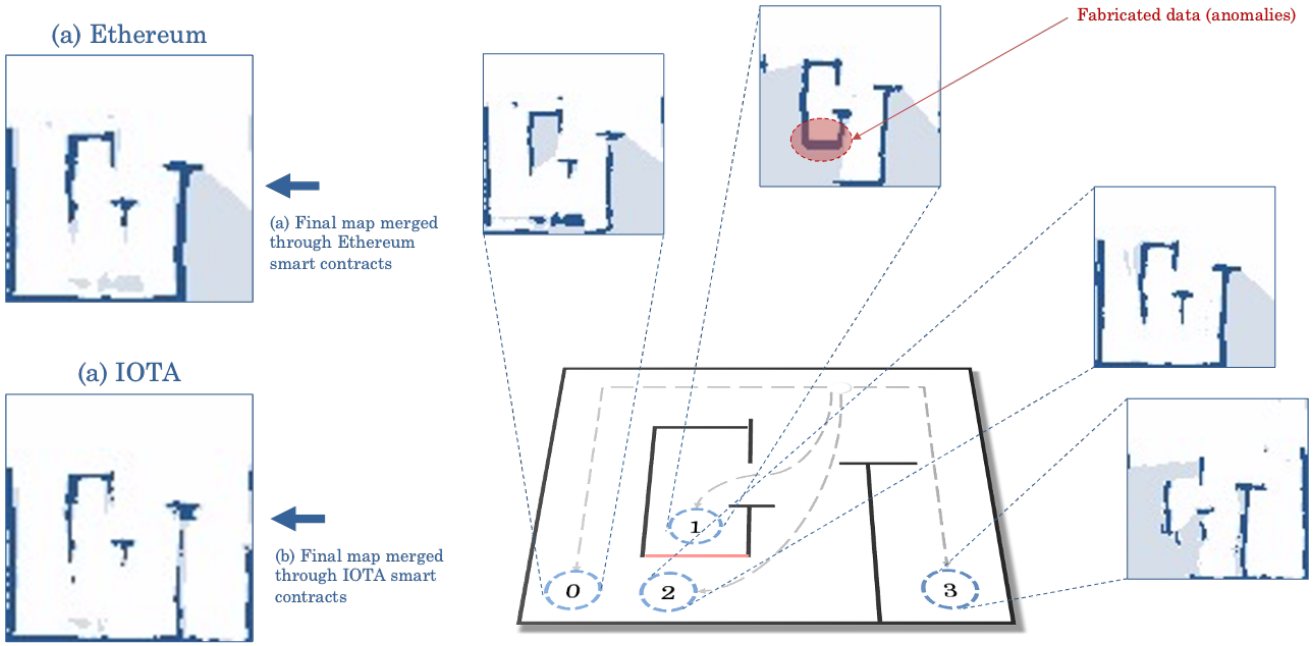


Fig. 9: Real-world maps merged by IOTA and Ethereum smart contracts.

TABLE III: Percentage of total area mapped by each individual robot, and percentage of total area generated by the Ethereum and IOTA smart contracts merging the individual maps.

Map generator	Mapped area (%)	Note
Robot 0	77.78 %	
Robot 1	65.55 %	*Byzantine agent*
Robot 2	83.95 %	
Robot 3	76.39 %	*Disconnected agent*
$\cup_{i=0}^{3} Robot_i$	94.64 %	
Ethereum SC	82.2. %	*Agents 0 + 2*
IOTA SC	95.50 %	*Agents 0 + 2 + 3*

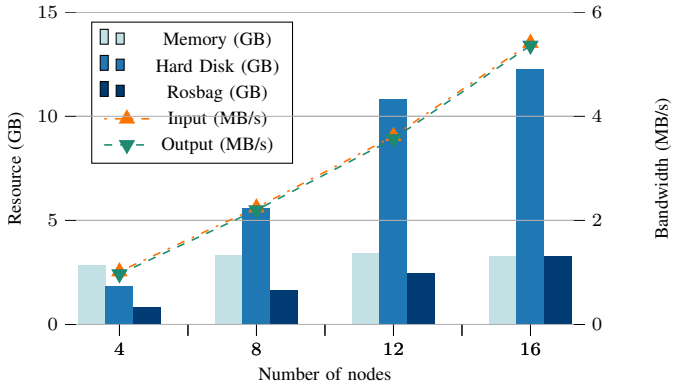


Fig. 10: Average memory consumption and hard disk usage of robots for 4, 8, 12, and 16 robots publishing lidar scan data on the IOTA ledger for one hour compared with the ROS bag of the lidar scan data. In addition, the average input and output bandwidth used by robots are measured in each setup.

with the number of nodes, the required bandwidth (almost 6 MB/s) and hard disk space (16 GB) are arguably reasonable for a group of 16 nodes and are within the same order of magnitude of cloud-based data recording solutions based on *ROS bags*. Additionally, it is worth mentioning that in this experiment raw lidar data is written on the ledger. Storage requirements can be optimized easily by substituting the raw data for hashes. This would also lead to reduced bandwidth usage. Additionally, IOTA allows snapshots of its ledger. After every mission, robots can store snapshots of the ledger. By doing this, robots do not need to store the whole ledger history from the beginning of their operation, and the encrypted data can be stored remotely in a cloud provider.

VI. Discussion

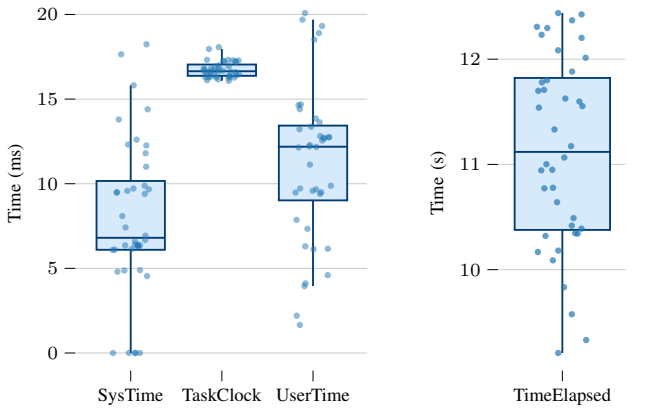
Throughout the results section we have shown the potential for the integration of IOTA in realistic missions in both simulation and experiments with ground robots. However, integrating blockchain technology naturally raises concerns in terms of computational overhead and system robustness, among other challenges. This section addresses these critical aspects.

A. Blockchain Overhead

In addition to the results above, we measure the CPU utilization time and the confirmation time for one of the smart contract methods. Figure 11 shows these results. The CPU utilization time indicates the actual impact of the IOTA smart contract in terms of resource utilization. Using the Linux perf tool, we measure the execution time of the *SubmitVotes* method of the Controller Swarm smart contract. Figure 11a shows that the typical execution time is in the range of 10 ms to 20 ms, revealing the potential for scalability and the ability of running multiple smart contracts in parallel. The use of DLTs and the decentralized consensus mechanisms in them, however, introduces a certain delay until a transaction is *confirmed*. In practice, this means that the transaction, in this

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3257984



(a) CPU utilization time of the $SubmitVotes$ method measured with the Linux perf tool.

(b) Confirmation time for the $SubmitVotes$ method.

Fig. 11: Distribution of the execution and confirmation time for the $SubmitVotes$ method of the CONTROLLER SWARM smart contract for each robot over multiple experimental runs. The actual CPU utilization time is in the range of a few *ms*, while the default IOTA network configuration results in confirmation times slightly over 10 s.

case the result of a smart contract method, is confirmed by the nodes in the Wasp Committee. Even if the actual network delay is small in the experiments, the default configuration of the IOTA research network introduces two delays of about 5 s each to avoid double spending. This results in confirmation times over 10 s reported in Fig. 11b. If a faster solution is required, the network configuration can be changed to reduce the confirmation time, given that the maximum network delay is known (e.g., within a single Wasp Committee that always maintains local connectivity or within a committee of nodes that uses wired connectivity, such as a network of charging stations). It is worth noting that this is a research network where functionality takes precedence over performance. In any case, the approach proposed in this paper is already valid for scenarios where consensus is not required in real-time but low impact on the use of computational resources is preferred.

In summary, we can assert that DLTs have potential to bring new standards of security and trust to large-scale distributed robotic systems.

B. Robustness: unreliable data and connectivity

We now shift the discussion towards two dimensions of the system robustness that are relevant for this manuscript: robustness in terms of data reliability and robustness in terms of connectivity reliability. In the Gazebo simulation, as illustrated in Fig. 6 the fabricated data colored as red is being removed from the final merged map. The same procedure happens in the other experiment shown in Fig. 9. These merging procedure shows an example of how the system is robust against unreliable data. As long as the malicious nodes do not constitute more than one third of a Wasp committee, the smart contracts' can compensate the errors introduced by unreliable data. Additionally, we achieve robustness against unreliable connectivity by dividing the smart contracts into CONTROLLER SWARM and AGENT SWARM contracts. In the Gazebo simulation experiment, even though the robots are divided into two network partitions, the final merged map has all the data, aggregated from robots in both groups. Also, In Fig. 9 we can see the artifacts of unreliable connectivity in the results from Ethereum smart contracts, while the IOTA-based solution is robust. Table III shows this as quantitative format.

C. Challenges and prospects

In addition, we have manifested that porting decision-making tasks on blockchain will not consume extra power compared to usual robotic tasks such as mapping. Despite all these benefits, there is a series of challenges that can be the objective of further investigation. First, there should be prior knowledge about the possible network splits since the disconnected nodes should still operate on the same chain. Second, this framework can only be applied if there are enough nodes in the network to form different chains. A third challenge is the limited capability of smart contracts in executing heavy decision-making tasks like machine learning based solutions. In future work, we are investigating how this kind of heavy process can be executed off-ledger. Fourth, based on the current implementations, we are working to report a thorough analysis of the performance of Wasp and GoShimmer nodes in robotic applications communicating with ROS 2. Finally, we have noted that the current version of IOTA supporting smart contracts is a research network where functionality is sought out over performance. Theoretically, the network design implies a more scalable solutions. However, there is a limited amount of real-world deployments. In any case, our experiments prove the effectiveness for many applications that do not require fast consensus for the smart contracts within the Wasp committees, but where data is already fed in real-time to the GoShimmer layer.

VII. CONCLUSION

This paper proposes a framework for partition-tolerant decision-making processes in multi-robot systems. Importantly, we propose a novel design approach to deploying smart contracts that incorporate the mentioned constraints. In general words, the logic is split into two IOTA smart contracts, which tolerate network partitioning at the transaction layer but not directly at the smart contract layer. Furthermore, we demonstrate how IOTA's two-layer structure and ROS 2 can be applied to a multi-robot system, with a study on the design approaches to follow and a novel design and implementation. In order to demonstrate the partition-tolerant framework and the proposed architecture, we chose a distributed mapping problem for a proof of concept. The distributed mapping task was first simulated in the Gazebo with eight robots. We then tested our distributed mapping smart contracts in a real-world scenario and compared the results to a baseline Ethereum implementation. Our results demonstrate how network partitioning impacts distributed decision-making outcomes in both cases. Only our IOTA-based implementation showcases both byzantine-tolerant and partition-tolerant behaviour. We also discuss the robustness of the system against unreliable data and unreliable connectivity, as well as in terms of byzantine tolerance.

Future research will be directed towards more diverse applications and large-scale experiments, particularly in integrating deep learning to smart contracts.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3257984

ACKNOWLEDGMENT

This research work is supported by the R3Swarms project funded by the Secure Systems Research Center (SSRC), Technology Innovation Institute (TII).

REFERENCES

[1] Melanie Schranz, Martina Umlauft, Micha Sende, and Wilfried Elmenreich. Swarm robotic behaviors and current applications. *Frontiers in Robotics and AI*, 7, 2020.
[2] Mohd Javaid, Abid Haleem, Ravi Pratap Singh, and Rajiv Suman. Substantial capabilities of robotics in enhancing industry 4.0 implementation. *Cognitive Robotics*, 1, 2021.
[3] Jorge Peña Queralta, Jussi Taipalmaa, Bilge Can Pullinen, Victor Kathan Sarker, Tuan Nguyen Gia, Hannu Tenhunen, Moncef Gabbouj, Jenni Raitoharju, and Tomi Westerlund. Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision. *Ieee Access*, 8, 2020.
[4] Olimpiya Saha and Prithviraj Dasgupta. A comprehensive survey of recent trends in cloud robotics architectures and applications. *Robotics*, 7(3), 2018.
[5] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1), 2013.
[6] Reza Olfati-Saber and Richard M Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on automatic control*, 49(9), 2004.
[7] Ruffin James White-Magner. *Usable Security and Verification for Distributed Robotic Systems*. PhD thesis, UC San Diego, 2021.
[8] Víctor Mayoral-Vilches. Robot hacking manual (rhm). *arXiv preprint arXiv:2203.04765*, 2022.
[9] Jorge Peña Queralta, Li Qingqing, Zhuo Zou, and Tomi Westerlund. Enhancing autonomy with blockchain and multi-access edge computing in distributed robotic systems. In *FMEC*. IEEE, 2020.
[10] Anum Nawaz, Jorge Peña Queralta, Jixin Guan, Muhammad Awais, Tuan Nguyen Gia, Ali Kashif Bashir, Haibin Kan, and Tomi Westerlund. Edge computing to secure iot data ownership and trade with the ethereum blockchain. *Sensors*, 20(14), 2020.
[11] Jorge Peña Queralta and Tomi Westerlund. Blockchain for mobile edge computing: Consensus mechanisms and scalability. In *Mobile Edge Computing*. Springer, 2021.
[12] Volker Strobel, Eduardo Castelló Ferrer, and Marco Dorigo. Blockchain technology secures robot swarms: A comparison of consensus protocols and their resilience to byzantine robots. *Frontiers in Robotics and AI*, 7, 2020.
[13] Eduardo Castelló Ferrer, Ernesto Jiménez, Jose Luis Lopez-Presa, and Javier Martín-Rueda. Following leaders in byzantine multirobot systems by using blockchain technology. *IEEE Trans. on Robotics*, 38(2), 2021.
[14] Mário Gabriel Santos de Campos, Caroline PC Chanel, Corentin Chauffaut, and Jérôme Lacan. Towards a blockchain-based multi-uav surveillance system. *Frontiers in Robotics and AI*, 2021.
[15] G. W. Clark *et al.* Cybersecurity issues in robotics. In *CogSIMA*, 2017.
[16] Jean-Paul A Yaacoub, Hassan N Noura, Ola Salman, and Ali Chehab. Robotics cyber security: Vulnerabilities, attacks, countermeasures, and recommendations. *International Journal of Information Security*, 2021.
[17] Fiona Higgins, Allan Tomlinson, and Keith M Martin. Threats to the swarm: Security considerations for swarm robotics. *International Journal on Advances in Security*, 2(2&3), 2009.
[18] Fiona Higgins, Allan Tomlinson, and Keith M Martin. Survey on security challenges for swarm robotics. In *2009 Fifth International Conference on Autonomic and Autonomous Systems*. IEEE, 2009.
[19] Alan G Millard, Jon Timmis, and Alan FT Winfield. Towards exogenous fault detection in swarm robotic systems. In *Conference towards Autonomous Robotic Systems*. Springer, 2013.
[20] Ian Sargeant and Allan Tomlinson. Modelling malicious entities in a robotic swarm. In *2013 IEEE/AIAA 32nd Digital Avionics Systems Conference (DASC)*. IEEE, 2013.
[21] Igor Zikratov, Oleg Maslennikov, Ilya Lebedev, Aleksandr Ometov, and Sergey Andreev. Dynamic trust management framework for robotic multi-agent systems. In *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*. Springer, 2016.
[22] David Saldana, Amanda Prorok, Shreyas Sundaram, Mario FM Campos, and Vijay Kumar. Resilient consensus for time-varying networks of dynamic agents. In *American control conference*. IEEE, 2017.
[23] Eduardo Castelló Ferrer. The blockchain: a new framework for robotic swarm systems. In *Future technologies conference*. Springer, 2018.
[24] Ilya Afanasyev, Alexander Kolotov, Ruslan Rezin, Konstantin Danilov, Alexey Kashevnik, and Vladimir Jotsov. Blockchain solutions for multi-agent robotic systems: Related work and open questions. *arXiv preprint arXiv:1903.11041*, 2019.
[25] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014), 2014.
[26] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, 2018.
[27] Serguei Popov. The tangle. *White paper*, 1(3), 2018.
[28] Salma Salimi, Jorge Peña Queralta, and Tomi Westerlund. Towards managing industrial robot fleets with hyperledger fabric blockchain and ros 2. *arXiv e-prints*, 2022.
[29] Salma Salimi, Paola Torrico Morón, Jorge Peña Queralta, and Tomi Westerlund. Secure heterogeneous multi-robot collaboration and docking with hyperledger fabric blockchain. *arXiv preprint*, 2022.
[30] Jason A Tran, Gowri Sankar Ramachandran, Palash M Shah, Claudiu B Danilov, Rodolfo A Santiago, and Bhaskar Krishnamachari. Swarmdag: A partition tolerant distributed ledger protocol for swarm robotics. *Ledger*, 4(Supp 1), 2019.
[31] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, 2008.
[32] Vasco Lopes, Nuno Pereira, and Luís A Alexandre. Robot workspace monitoring using a blockchain-based 3d vision approach. In *EEE/CVF CVPR Workshops*, 2019.
[33] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3. Kobe, Japan, 2009.
[34] Eric A Brewer. Towards robust distributed systems. In *PODC*, volume 7. Portland, OR, 2000.
[35] Evaldas Drasutis. Iota smart contracts. Technical report, IOTA Foundation, 2021.
[36] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016.

Farhad Keramat received his B.S. degree in Electrical Engineering and his M.Sc. degree in Secure Communication and Cryptography, from University of Tehran, Tehran, Iran, in 2017 and 2020, respectively. Since 2021, he has been a researcher at the Turku Intelligent Embedded and Robotic Systems (TIERS) Group, University of Turku. His research interests include distributed ledger technologies, multi-robot systems security and multi-robot collaboration.

Jorge Peña Queralta is a postdoctoral researcher at the Turku Intelligent Embedded and Robotic Systems (TIERS) Group, University of Turku, Finland. He received B.Sc. degrees in mathematics and physics engineering from UPC BarcelonaTech, Spain, in 2016, a M.Sc. (Tech.) degree in ICT from the University of Turku, a M. Eng. degree in Electronics and Communication Engineering from Fudan University, China, in 2018, and a Ph.D. (Tech.) degree from the University of Turku in 2022. His research interests include multi-robot systems, collaborative autonomy, distributed perception, and robot learning.

Tomi Westerlund is a Professor of Autonomous Systems and Robotics at the University of Turku and a Research Professor at Wuxi Institute of Fudan University, Wuxi, China. Dr. Westerlund leads the Turku Intelligent Embedded and Robotic Systems research group (tiers.utu.fi), University of Turku, Finland. His current research interest is in the areas of Industrial IoT, smart cities and autonomous vehicles (aerial, ground and surface) as well as (co-)robots. In all these application areas, the core research interests are in multi-robot systems, collaborative sensing, interoperability, fog and edge computing, and edge AI.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/