ISRaft Consensus Algorithm for Autonomous Units

Linir Zamir, Aman Shaan and Mehrdad Nojoumian
Department of Electrical Engineering and Computer Science
Florida Atlantic University
Boca Raton, FL
{lzamir2016, ashaan2019, mnojoumian}@fau.edu

***Abstract*—Consensus protocols are a key feature in decentralized systems where multiple unreliable nodes operate, e.g., in Blockchain technologies with many worldwide applications such as supply chain management, cryptocurrencies and information sharing. ISRaft is a consensus protocol built upon Raft, a previously developed protocol that is used for replicated state machines when a group of nodes is required to achieve a consensus related to the state of the machine. This paper therefore proposes an alternative version of the ISRaft consensus protocol to allow communication among nodes in a secured fashion while maintaining the security features of the original ISRaft algorithm even in the presence of adversarial nodes. The proposed model utilizes a trust parameter to enforce cooperation, i.e., a trust value is assigned to each node to prevent malicious activities over time. This is a practical solution for autonomous units with resource-constrained devices where a regular encrypted communication method can negatively affect the system performance.**

***Index Terms*—Consensus Algorithm, Autonomous System, Blockchain Technology, Trust Model, Raft**

I. Introduction

Autonomous units can operate under many different operations and models. Much like a human, each unit is required to make decisions and be able to communicate and act upon it. Comprising many small individual nodes, autonomous units in decentralized fashion have no centralized authority to guide them. Instead, they rely on individual communications in order to complete tasks, achieve consensus and share information [1]. These groups are robust, flexible and scalable, which allows them to complete many different tasks in different fields. Decentralized autonomous units contain many individual nodes with several advantages. The loss of a few nodes will not affect the system, contributing to its robustness. Flexibility is possible because of simple communications that take place between nodes, and scalability works since this method of operation can work together with different numbers. Autonomous units can be applied in fields that deal with data collection and management. Some examples are exploration, surveillance, and data classification. The methodologies of such systems are efficient by themselves, but when implemented with other blockchain technologies, many new opportunities arise.

The Blockchain infrastructure was introduced in 2009 through the creation of Bitcoin. Bitcoin required the nodes of a system to come to a consensus in order to append blocks to the blockchain. In most blockchain systems, the *Proof-of-Work* (PoW) consensus protocol is used. However, this protocol is known to be slow and energy consuming, which makes it impractical for many applications. Blockchains are being established in many fields of technology in order to improve security and provide features such as decentralization. One of these applications is in autonomous driving [2], where security is extremely important in order to establish autonomous vehicles that are reliable and safe. Blockchain technology has also extended to Robot Swarms. The presence of a blockchain in robot swarms allows for these nodes to have easier and more efficient communication. Blockchains allow robot swarms to have access to shared knowledge while still retaining the individual communications that take place in swarms. Furthermore, the immutable characteristics of the blockchain also enable robot swarms to be more secure and reliable in terms of data storing and sharing.

This paper therefore proposes an alternative version of the *Information Sharing Raft* (ISRaft) consensus protocol [3] to allow communication among nodes in a secured fashion while maintaining the security features of the original ISRaft algorithm even in the presence of adversarial nodes. ISRaft builds upon the Raft consensus protocol that achieves consensus through the process of leader elections, voting, block generating and validating. ISRaft differs from the original Raft by adding Byzantine Fault Tolerance, allowing all nodes in a network to request data changes, validation of any data types, and adding reputation value to each node. In ISRaft, individual nodes in the system are able to communicate with each other through *Remote Procedure Calls* (RPC).

The proposed model utilizes a new trust parameter to enforce cooperation [4], i.e., a trust value is assigned to individual nodes to prevent malicious activities over time. This is a practical solution for autonomous units with resource-constrained devices where a regular encrypted communication method can negatively affect the performance of the entire system. This modification intends to further improve the security and capabilities of ISRaft. In the original Raft, any node is able to become the leader of the particular system, allowing potential malicious nodes to compromise the system if elected as a leader. Through the use of ISRaft, the new trust parameter prevents malicious nodes from becoming leaders and thereby preventing them from compromising the system. We also implement ISRaft into a robot swarm system on ARGoS in order to test its efficiency. Each robot in the swarm becomes its own node with its own trust value.

978-1-6654-4131-5/21/$31.00 ©2021 IEEE

The major goal of this research is to solve the problem of secured and immutable communication in P2P networks by using a decentralized consensus protocol solution. Consensus protocols, such as PoW, use mining procedure as a way to guarantee the soundness of the block, where miners are also compensated for their works; see [5] for details of Bitcoin mining in Blockchain. In ISRaft consensus protocol, the node who mines the block is the *leader* of the cluster, who is elected based on its trust value between himself and other nodes in the cluster. In this protocol, trust is represented as a resource with a numeric value. Nodes can use this parameter to increase their chances of becoming leaders, and to generally communicate with other nodes in the cluster. Unlike other consensus protocols, ISRaft can operate independently and does not require solving a complex mathematical puzzle, which makes it cost-efficient. Furthermore, ISRaft is simple enough to run on autonomous units with minimum computational power while maintaining consensus, which makes it ideal for several applications with resource-constrained devices.

II. Related Works

In this section, we discuss studies that proposed blockchain-based mechanisms in different autonomous units and how our implementation is different from previous works.

Several autonomy-related projects [6]–[8] aim at finding a solution for reliable communication and decision-making among autonomous units because Byzantine units can have disastrous effects on any autonomous technology. A blockchain approach utilizing Proof-of-Work and other known protocols has been implemented for testing among autonomous agents. By using blockchain, data can be stored in a safe way to prevent unintended or malicious data changes by byzantine units present in the system. Autonomous vehicles, swarm robotics and smart devices are good examples of autonomous units that can benefit from the use of blockchain as a core part of their infrastructures.

Ferrer et al. [9] highlights the importance of swarm robotics and how the blockchain can be utilized to further improve robotic technologies. Swarm robotics can be seen as a kind of autonomous unit where the characteristics of swarms, such as scalability and resistance to failure, have made swarm robotics appealing to many researchers. However, swarm robotics have some drawbacks. Since swarms mainly communicate through other neighboring robots, there is nothing that allows them to have shared knowledge with other robots in the system. This is where the use of a blockchain is beneficial, where global knowledge can be added to swarm robotics while maintaining local knowledge. By implementing blockchain technologies in swarm robotics, the security of these swarms can also be improved. The blockchain provides reliable and safe communication among robots along with opportunities to validate members of a swarm and prevent malicious attacks. Blockchain also allows for distributed decision-making in swarms. Processes such as voting and leader elections can be implemented into swarms through decision-making that is made possible through the use of a blockchain.

Strobel and Dorigo [10] designs a robotic swarm that defines which color is more prevalent in an environment of both black-and-white tiles. The authors' goal is to determine the frequency of black tiles using a blockchain approach among robots, instead of simply finding which color is more prevalent. Furthermore, a reputation management system is added to manage the presence of byzantine robots. This utilizes a smart contract where each robot's reputation is stored on the blockchain. The reputation of each robot is changed based on how they report the color of a certain tile when the report is compared with other robots' reports in the swarm. When an individual robot reports an incorrect color, its reputation is decreased. When a robot's reputation value becomes very low, its votes are ignored. They then run three separate experiments. The first one is to test if the correct frequency of tiles could be calculated without using reputation management. The second experiment is to study the effects of the swarm on the blockchain and how efficient it will be. The third experiment is to determine if byzantine robots will have an effect while using the reputation management in the swarm. Overall, the results show that having reputation management is effective in handling byzantine robots at a smaller scale.

Singh et al. [11] present an improved way of decision-making process of swarm robotics by using a blockchain system. Currently, the ways that robots communicate are too centralized and are not resource efficient since blockchains require a large number of resources. In order to improve this, the authors propose to utilize *Proof-of-Authority* (PoA) instead of the more common PoW. In PoA, certain nodes are given the roles of validator. The validators manage and record transactions into the blockchain, with the validator leader gaining the block-publishing priority. This leader is changed after a set period. To verify this approach, a similar method to the colored tiles is used. In a simulated environment, there are grey, white, and black tiles. The robots emit red when on white tiles, green when on black tiles, and do not respond to grey tiles. The objective is to have the robots decide which color tiles emit each color of light. While this test does not consider the byzantine robots, the authors show that it is effective and less resource-intensive compared to the normal blockchain.

Queralta et al. [12] improve data sharing, resource utilization and communication in swarm robotics by using blockchain technologies. They use the PoW consensus protocol to measure the resources the system still has in order to provide proper resource utilization. Moreover, the *Proof-of-Stake* (PoS) is used to validate transactions due to the scalability issues when using PoW. They also use a Single Longevous Blockchain that utilizes ad hoc collaboration in order to allow new nodes to enter the system. This blockchain will also work in situations using permissioned or permissionless blockchains. In addition, a method of ranking is implemented by using smart contracts, which has resulted in certain robots having higher ranks than others. These rankings are changed based on the needs of the system and will not be recorded in the blockchain itself. Only the data that is validated is stored in the blockchain. Despite the presence of the ranking

system, the nodes are not chosen based on their rankings or levels of trust, which provides protection for malicious nodes.

The implementation of blockchain among autonomous units provides many benefits as shown in the aforementioned papers. However, in most cases, malicious nodes are not considered. Furthermore, in the majority of these papers, proof-of-work is utilized that is not efficient enough. For instance, Strobel et al. [13] propose a method of managing these byzantine nodes through a reputation value and utilization of proof-of-work. However, if a malicious node has a high hash-rate, he can still compromise the system.

We intend to address these issues by utilizing the Raft consensus protocol. Due to the nature of this protocol and its periodical leader elections, malicious nodes will not be able to compromise the system regardless of their computational powers unless they become a leader. To further prevent a malicious node from becoming a leader, we utilize a trust parameter that allows the reputable nodes to become leaders. Our framework works in both permissioned and non-permissioned blockchains. Finally, we utilize the idea of RPC in the leader election process that is similar to smart contracts.

III. ISRaft Consensus Protocol

In this section, we present the ISRaft consensus protocol in a closed and isolated network. ISRaft protocol uses similar fundamentals as the original Raft algorithm, i.e, Leader Election and Server Communication. In addition to modifying these features, this new design includes a trust value, message validation and security features in the presence of an adversary.

A. Design Overview

The main idea of ISRaft protocol is to have all nodes in the cluster to hold an agreed upon ledger, containing messages of communication among different nodes. A node in the cluster can be in one of the three states: *follower, candidate* or *leader*. A leader is the only authority in the cluster that can append blocks to the chain, making it the miner of the current block. The other nodes can send messages to each other and the leader, and the leader then has the responsibility to distribute the message among all other nodes. For simplicity, communication among nodes in the network is done via RPC messages for minimum coding and faster initialization.

1) **Asymmetric Encryption:** Each node is initialized with a pair of cryptographic keys used for authentication and signature. The public-key is submitted and logged into a registry that holds all identities in the network. Other nodes will receive an update whenever a new node joins the network. Whenever a node sends any RPC message, he signs it with his own private-key. Nodes reject any RPC message that does not include a valid signature. Any key pair, such as RSA, can be used for the purpose of this model. This also helps with tracing back messages when a trust is broken between nodes.
2) **Permissioned Blockchain**: New nodes need to go through an enrollment process where a key pair is initiated and stored on the registry along with its address, trust value, and other required parameters.
3) **Secured Channel:** For simplicity, we assume that the communication of the nodes is done on a secured channel, without the possibility of having a man-in-the-middle attack. This, however, can be easily addressed in future works. Furthermore, communication is done via RPC messages, signed by each node.

As mentioned, a node in the system can be in either one of the following identities: *follower, candidate* or *leader*.

- **Follower:** This state is the initial state of a node upon joining the network. While in this state, the follower operates under the current leader of the cluster. If a follower node wishes to add anything to the chain, the leader needs to first validate the message, and only then it can be committed. A follower receives a stream of heartbeat messages from the leader to make sure it is updated to the last block. A heartbeat message is sent every short period of time, called *election timeout*, and it can include new transactions, data and/or leader change.
- **Candidate:** When a follower does not receive heartbeat messages within some predetermined period of time, it automatically changes its state to a candidate. While in this state, the node sends a request-vote message to all other nodes in the cluster, asking for their vote so that he can become the leader, and so the miner of the next block. A candidate can not vote for himself.
- **Leader:** A candidate can become a leader when he receives at least $n/2+1$ votes in a cluster of n nodes. A leader has the responsibility to mine the next block, and to send heartbeat messages to all nodes in the network.

Trust between two nodes plays an important factor in this ISRaft protocol. Trust represents the confidence in which nodes rely on each other. Let P_i be a node with a public-key identifier i. P_i then holds a trust value associated with each node in the network. The trust value assigned by P_j to P_i is then T_i^j. This value is in the interval $[0, 1]$. These values are unique and do not have to be symmetric, i.e., $T_i^j \neq T_j^i$ [14]. When a node sends an RPC message of any kind (excluding heartbeat message and responds) to the other nodes in the network, the recipients immediately decrease the trust value of the sender by a fixed small amount. This is done as a way to reduce the total number of messages that can be sent every single second, decreasing the possibility of overflowing and DDoS attacks on the network by any node in the system.

ISRaft uses the following RPCs for its operation:

- *RequestVote* - This RPC is sent whenever a *candidate* is calling for votes.
- *RequestAdd* - Every node in the cluster can send this message to the leader, asking for some data to be added to the next generated block.
- *AppendBlock* - The leader can send this message whenever a new block is mined. This can happen every fixed time, or by the decision of the leader.

- *ApproveCommit* - After a new block has been added to the node, he sends this RPC back to the leader.

These simple RPCs are what makes this consensus protocol easy to implement and operate in resource-constrained devices.

B. Leader Election

For this protocol to work, a leader must be elected. This process is done automatically when no heartbeat message is being sent over *election timeout*. After that period of time, the *follower* node changes its state to a *candidate* and he begins the election process. During this time, the node sends *RequestVote* RPC concatenate with the latest block number signed with his private-key signature to all nodes in the cluster. A follower or a candidate node that receives the RPC will then send a vote if and only if the following conditions are met:

- The node has not received any heartbeat messages from the current leader.
- The latest block number is at least equal to the current node's term plus 1.
- The *RequestVote* RPC is signed with a valid candidate's private-key.
- The trust value T of the sender is at least 0.5.

A node that receives the first *RequestVote* RPC will hold the first vote until the end of the *election timeout* regardless of the candidate's trust value, as long as the candidate fulfills the conditions. However, in the case where more than one *RequestVote* arrives, the node will choose who to vote for based on the trust value. A candidate with a higher trust value T will have a higher chance of getting the vote. A leader is elected when he receives the votes of the majority of the cluster. At that point, the elected leader sends out an RPC that includes the signed vote messages that he received from the nodes in the cluster. This acts as a proof of election and also prevents a self-promoted leader. All other nodes in the cluster then increase the trust value of the appointed leader by some fixed amount. This can be seen as a reward given to the elected leader who will then mine the next block.

After the new leader has been elected, he starts sending heartbeat messages containing the signatures of all other nodes who voted for him as a continuous proof of his leadership. If a new node was not aware that a new leader had been elected, the heartbeat message serves as an update for the node to follow. A leader is elected for a limited amount of time, called *leadership term*. At the end of this term, the leader gets to mine a new single block added to the chain and then a new election process begins.

During the election process, a node can receive a heartbeat RPC from a different node claiming to be the new leader. If the block number of the new heartbeat is at least higher than the block number of the previous heartbeat message, the node will recognize that new node as the leader and will follow the new leader. Otherwise, it will reject the heartbeat RPC and will continue with the election process.

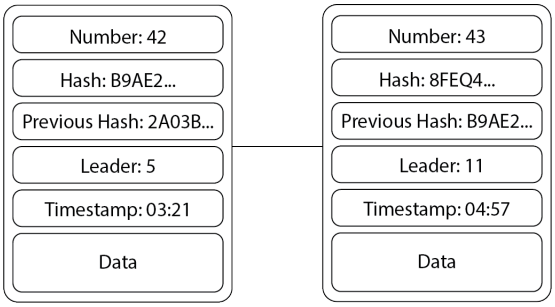


Fig. 1. Example of ISRaft blocks.

C. Block Generation

Every leadership term ends with a block being appended to the chain. A newly created block has the following data, as shown in Figure 1: block number, block hash, previous hash, block leader, timestamp, and data array.

When a follower node intends to add data to the block, he sends a request via the *RequestAdd* RPC. This RPC is signed by the requesting node. The signature guarantees the authenticity of the request, which prevents a malicious node from forging another node's request. This RPC is sent to all nodes in the cluster including the leader. This is performed for two purposes: To prevent manipulation of data, and to guarantee that a leader will receive the data in the case if a new leader has been elected without the node's knowledge. When the *leadership term* is coming to end, the leader begins the new block generation process by hashing the value of the new block and sending the *AppendBlock* RPC to all nodes in the cluster. This message contains the hash value of the block to be appended and all signed votes from the majority of the nodes in the cluster. When a node receives the *AppendBlock* RPC, he performs the following:

1) Compares the hash in the RPC message to make sure it is the expected hash value,
2) Verifies if the block number is larger than the last committed block, and
3) Validates if the leader's votes are legitimate.

If all conditions are satisfied, the node increases the trust value of the leader and sends a signed *ApproveCommit* RPC to the leader. If the majority of nodes in the network send this approval message, the leader can then send a second RPC that includes the signed approvals of all nodes in the network. When a node receives the second RPC, it commits the new block to the chain.

D. Data Update and Validation

Nodes are able to add data to the block through a request made to the leader of the current term. The leader then decides to store the information received from the nodes into the next block. The data is locally stored in a 2D array on each node, where the first dimension is the number of nodes in the network and the second dimension is the decision made by each one. The decision of each node will be visible to others and will be used to modify the trust values of other nodes.

Every task is bound in time, and during that time, it is expected that the system will achieve a consensus. Examples for such tasks can be learning the color of the surface, reading texts, or identifying threats. When a node makes a decision and commits it to the blockchain, other nodes can read that decision and change the trust value of that node accordingly. For example, if the system needs to identify the color of the surface, each one of the nodes will read the color and submit its decision to the current leader, who will in turn commit these decisions into the next block. At the end of the task, the last block can be read to make a final decision and validate it. The validation process heavily relies on the trust values of nodes in the network. When a consensus is required, the reputation value of each node is calculated based on individual trust values, which is defined as follows [15]:

Definition 1: Let T_i^j be the trust value assigned by P_j to P_i. Let T_i be the reputation function that illustrates how trustworthy P_i is:

$$T_i = \frac{1}{n-1} \sum_{j \neq i}^{n} T_i^j$$

At the end of a task, all reputation values are calculated and are used as the weighted value for each node's decision. This can be written as $D = \sum_{i=1}^{n} T_i \cdot P_i^d$, for decision D, n number of nodes, and P_i^d as the decision of single node i.

IV. Experiment

A. ARGoS Simulator

For the purpose of testing and evaluating ISRaft consensus protocol, the ARGoS simulator was used [16]. ARGoS is a swarm robotic simulator that is able to simulate large-scale swarms for any purpose. In our research, each robot acts as an independent ISRaft node, operating under the same guidelines detailed in the previous sections. Each node was given access to a running ISRaft network and was able to read and receive commands by having a unique identifier in the network.

A single node was able to execute commands either willingly (such as the heartbeat messages) or through a client controller running simultaneously. This was used for testing malicious activities such as message droppings, or messages that were not part of the original protocol.

The experiment was conducted as a simulated computer cluster with similar hardware to small mobile phones - a single core with 1.5GHz and 2 GB of memory. A total of 20 nodes operate with the ARGoS Simulator where the output data was carefully monitored by a 3rd party operator.

B. Setup

A total of 20 nodes were used in the experiment similar to [13]. The goal of this experiment was to help a set of autonomous units to make decisions about the colors of tiles in a 20×20 grid of black, white and gray tiles. The colors of these 400 tiles were randomly selected. For simplicity purposes, the nodes were able to communicate with each other freely, without distance restriction.

At the beginning of the experiment, each node was set to hold the data type that is relevant for this specific experiment. A 2D array was initialized as `char arr[20][400]` where the first dimension was the number of nodes in the cluster and the second dimension was the number of tiles in the system.

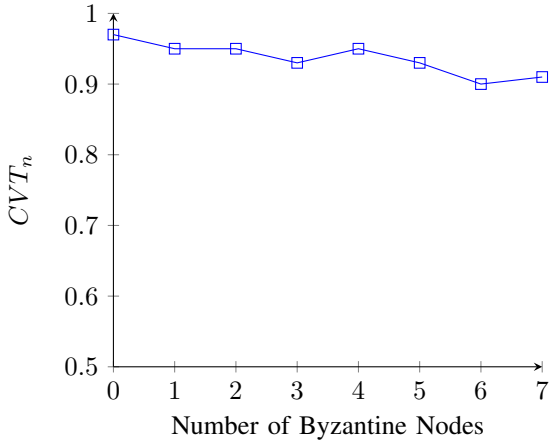
The experiment was set as follows: when a node stands on a colored tile, it reads that color. Upon success, the node then sends the signed *RequestAdd* RPC to the cluster including the tile number (1 to 400) and the color it reads. The leader receiving the RPC will then change the data of the 2D array at the location of the requesting node with the tile number and color reported. After some time, the leader will then send the *AppendBlock* RPC to the cluster with the new changes. A node receiving the RPC will then check to see if it has already read the color of that same tile. If yes, he will compare the results. If it is a correct color, the trust value of the reporting node will be increased, otherwise, it will be decreased. At the end of the experiment, the reputation value for each node is calculated and the decision for each tile is made based on the reputation values and reported colors. If, for example, tile 4 had 2 votes: a node with reputation 0.45 voted "Black" and a node with reputation 0.8 voted "White," the final decision will be "White."

During the experiment, Byzantine nodes were added to the cluster to see how the trust values are changed based on false messages. These Byzantine nodes always report wrong colors.

C. Technical Results

The experiment ran 35 times with different *election timeout* values ranging from 1 to 100 millisecond (assigned randomly) for a total of 5 minutes for each experiment. At the end of each run, we calculated the percentage of correctly voted tiles, noted as CVT_n for experiment n.

For each experiment with the same number of byzantine nodes, we averaged the CVT value to get these results:



As shown above, despite the decrease in CVT value when more byzantine nodes were added to the experiment, the overall value never went below 90%. This illustrates that the validation process along with the calculated trust values help in maintaining correctness of the model.

V. Conclusions and Future Work

ISRaft is a consensus protocol that can be implemented among autonomous units in a secure data sharing environment. This protocol makes it possible to achieve consensus in the presence of adversarial nodes. Moreover, by using trust and reputation values, it becomes possible to validate the authenticity and correctness of the shared data. This can be easily implemented on resource-constrained devices and our model works perfectly in any infrastructure where regular encrypted communication methods can negatively affect the performance of the system.

In our future work, we will expand the implementation of ISRaft to deal with different tasks. We will also implement the protocol on real autonomous units to see how it operates in real-life scenarios.

VI. Acknowledgment

Research was sponsored by the Army Research Office and was accomplished under Grant Number W911NF-18-1-0483. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intelligence*, vol. 7, no. 1, pp. 1–41, 2013.

[2] Y. Yuan and F.-Y. Wang, "Towards blockchain-based intelligent transportation systems," in *IEEE 19th International Conference on Intelligent Transportation Systems*, 2016, pp. 2663–2668.

[3] L. Zamir and M. Nojoumian, "Information sharing in the presence of adversarial nodes using raft," in *Future Technologies Conference*, 2021.

[4] M. Nojoumian, "Rational trust modeling," in *International Conference on Decision and Game Theory for Security*. Springer, 2018, pp. 418–431.

[5] M. Nojoumian, A. Golchubian, L. Njilla, K. Kwiat, and C. Kamhoua, "Incentivizing blockchain miners to avoid dishonest mining strategies by a reputation-based paradigm," in *Computing Conference*. Springer, 2018, pp. 1118–1134.

[6] T. Hardjono, A. Lipton, and A. Pentland, "Toward an interoperability architecture for blockchain autonomous systems," *IEEE Transactions on Engineering Management*, vol. 67, no. 4, pp. 1298–1309, 2019.

[7] M. Baza, M. Nabil, N. Lasla, K. Fidan, M. Mahmoud, and M. Abdallah, "Blockchain-based firmware update scheme tailored for autonomous vehicles," in *IEEE Wireless Communications and Networking Conference*, 2019, pp. 1–7.

[8] Y. Wang, Z. Su, K. Zhang, and A. Benslimane, "Challenges and solutions in autonomous driving: A blockchain approach," *IEEE Network*, vol. 34, no. 4, pp. 218–226, 2020.

[9] E. C. Ferrer, "The blockchain: a new framework for robotic swarm systems," in *Future Technologies Conference*. Springer, 2018, pp. 1037–1058.

[10] V. Strobel and M. Dorigo, "Blockchain technology for robot swarms: A shared knowledge and reputation management system for collective estimation," in *11th International Conference on Swarm Intelligence*, vol. 11172. Springer, 2018, pp. 425–426.

[11] P. K. Singh, R. Singh, S. K. Nandi, K. Z. Ghafoor, D. B. Rawat, and S. Nandi, "An efficient blockchain-based approach for cooperative decision making in swarm robotics," *Internet Technology Letters*, vol. 3, no. 1, p. e140, 2020.

[12] J. P. Queralta and T. Westerlund, "Blockchain-powered collaboration in heterogeneous swarms of robots," *arXiv preprint arXiv:1912.01711*, 2019.

[13] V. Strobel, E. Castelló Ferrer, and M. Dorigo, "Managing byzantine robots via blockchain technology in a swarm robotics collective decision making scenario," in *International Conference on Autonomous Agents and MultiAgent Systems*. ACM, 2018, p. 541–549.

[14] M. Nojoumian and T. C. Lethbridge, "A new approach for the trust calculation in social networks," in *E-business and Telecommunication Networks: 3rd International Conference on E-Business, Best Papers*, ser. CCIS, vol. 9. Springer, 2008, pp. 64–77.

[15] M. Nojoumian and D. R. Stinson, "Social secret sharing in coud computing using a new trust function," in *10th IEEE Annual International Conference on Privacy, Security and Trust*, 2012, pp. 161–167.

[16] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle *et al.*, "Argos: a modular, parallel, multi-engine simulator for multi-robot systems," *Swarm intelligence*, vol. 6, no. 4, pp. 271–295, 2012.