Blockchain-Enabled Redundant Fractionated Spacecraft System

Mariana Alves, Justine Veirier D'Aiguebonne, Thibault Gateau and Jérôme Lacan
Institut Supérieur de l'Aéronautique et de l'Espace (ISAE-SUPAERO),
Université de Toulouse,
31055 Toulouse, FRANCE
{mariana.de-andrade-dias-alves, justine.veirier-d-aiguebonne}@student.isae-supaero.fr
{thibault.gateau, jerome.lacan}@isae-supaero.fr

***Abstract*— Services and applications provided by satellites are continuously improving in terms of quality and diversity, as well as, their complexity. Resilience of traditional monolithic spacecraft is achieved mainly through redundancy, which exponentially increases the complexity of their production, and therefore, their cost. One solution would be to use fragmented spacecraft systems. Redundancy is achieved by nature with local complexity. Furthermore, scalability of such a system is facilitated. However, data transfer and physical communication link between the "fragments" must be addressed. The main challenge of these systems is the management of the network of satellites. Traditional centralised networks, managed by a single entity, have been proven not to be secure, as they generate a common point of failure, susceptible to attacks. Traditional distributed networks, managed by multiple entities, require honest participants and trust between entities, restricting collaborative mission applications. In the present article, a design for a blockchain-enabled swarm fractionated system managed by multiple trustless entities is proposed. The system is composed of functionally different nano-satellites which perform tasks from different subsystems in order to replicate the functionalities of a monolithic spacecraft. The blockchain nodes are deployed in the satellites and allow sharing of sensor data between the network. A consensus protocol ensures the validity of the shared data. The proposed system has been implemented and evaluated on a local blockchain composed of four Ethereum nodes. We believe that the proposed application opens the way to new collaborative missions between entirely trustless parties, ensuring transparency and cooperation within the system.**

TABLE OF CONTENTS

1. INTRODUCTION

Aerospace and computer engineering advancements have allowed for miniaturisation in space systems. Sensing satellites, for example, with the size of a large postage stamp and a mass below 10 g, have been launched and tested in earth-orbit [1]. CubeSat standard [2] has significantly decreased production costs of nano-satellites, allowing smaller companies and universities to perform autonomous space system missions [3]. However, hardware limitations restrict data storage, processing power and transmission power in nano-satellites, making them less resilient and powerful than their larger counterparts.

Due to these limitations, multi-satellite or swarm systems have grown in popularity [4]. These systems offer added resilience through the significant number of satellites involved. Failure of a satellite in a swarm system does not compromise the system's mission and can be resolved by replacing the satellite [5]. Additionally, since swarm systems do not have a limited physical range, their performance in specified sensing missions can be better than the performance of monolithic satellites [6, 7]. Another appealing characteristic of swarm systems is their interchangeability. Because different satellites carry different payloads, it is possible to alter a given satellite in the system in order to perform a different mission [4]. Regarding accessibility in the space sector, swarm systems also allow smaller players in the sector to perform complex missions through collaborative projects. The main challenge in these missions becomes the management of the multi-entity system.

A common method to manage a network of satellites is the use of a central entity which directly manages the whole system and establishes an interface between the system and all involved entities. However centralisation of the system creates a point of failure, susceptible to attacks, and diminishes the advantages of using a multi-satellite system.

Significant research has been made regarding the implementation of distributed networks in swarm systems [6, 8–13]. In such distributed networks, cooperation between spacecrafts are necessary to accomplish missions. Cooperation involves data sharing. However, an important issue in space activities is the security in communications. Distributed systems are susceptible to several attacks, such as illegal eavesdropping on satellites, interception of missions through flooding of the system with information (Denial-of-Service attacks) or manipulation of results through feeding of false information, etc. Therefore, in collaborative applications, a distributed network tolerant to failures and attacks is required.

Due to its trustless nature, blockchain as been proposed as a viable peer-to-peer network for multi-satellite systems [4, 8, 14–22]. The blockchain network can be described as a decentralised immutable database organised as a chain of data blocks cryptographically linked. Its resilience is ensured by a fault-tolerant consensus mechanism which synchronises the data shared by all members of the network and manages the member's access to the network.

The objective of the current paper is to present **a new application for the blockchain network in a collaborative**

multi-entity system. The system is composed of functionally different spacecrafts which perform tasks pertaining to different subsystems of an equivalent monolithic spacecraft. Each spacecraft embeds a node of the blockchain network and data from every payload is shared by all members. **Physical resilience** is ensured by **the swarm**, while **security resilience** is ensured by **the blockchain**. The proposed application opens the way to new collaborative missions between entirely trustless parties, ensuring transparency and cooperation within the system.

The document is organised as follows. In the next Section, the main concepts of blockchain and swarm systems are presented. In Section 3, the paper objectives are recalled and the developed system is presented. In Section 4, a practical scenario is presented with its implementation choices. In Section 5, results of the implemented network are discussed. In the final Section, the conclusion to the paper is presented and future work is detailed.

2. DEFINITIONS AND STATE OF THE ART

In this section, the definitions of satellite swarm, followed by blockchain are presented. After each definition, the state of the art of the concept is presented.

Satellite Swarms

Multi-satellite missions, such as swarm missions, are defined by NASA GSFC as follows: "An end-to-end system including two or more space vehicles and a cooperative infrastructure for science measurement, data acquisition, processing, analysis, and distribution." [10] where the system's vehicles are also known as agents [12].

Although there is no consensus surrounding the definition of swarm mission, their fundamental properties are (1) distribution, as tasks and data are shared amongst agents; (2) interdependency between agents, as missions depend on cooperation between agents; and (3) autonomy [23, 24]. Sterritt *et al.* [24] define the general objectives of an autonomic system as self-configuration, self-healing, self-optimisation and self-protection and the general attributes as self-awareness, self-situation (environment & context awareness), self-monitoring and self-adjusting.

Regarding distributed space systems, most research differentiates constellations and formation flight satellites from swarms, as the latter usually do not maintain a tight geometric formation [10, 25, 26]. However, Engelen et al. [5] note that navigation-wise, the requirements for a spacecraft swarm can be as strict or even more strict than a formation flying mission, due to the dynamics of the system. The swarm changes more rapidly than a fixed formation flight, and it is therefore more important to have accurate navigation information per element. Consequently, in the following work, the terms "formation flight system" and "cluster" will be considered as a synonym of a "swarm".

A differentiating characteristic of swarm systems, when compared to other multi-satellite systems, is that they involve a large number of flight units [25]. The concept of swarm is, therefore, said to focus on risk mitigation by fault-acceptance, rather than fault tolerance. In the case of a satellite swarm, risk mitigation is usually handled by the sheer number of spacecraft, as the loss of a single element will not result in a catastrophic failure [5] and can be resolved by the replacement of the agent.

Fractionated Spacecraft— Fractionated Spacecraft Systems can be seen as a subset of swarm systems as they allocate different mission functionalities to multiple flight units [25, 27]. The idea behind this technology is to decompose the traditional integrated monolithic satellite into several functioning modules. Its main advantages are, similarly to swarm systems [28] :

- the reduction of costs associated with production. The independent functional modules used in fractionated spacecraft can be mass produced and used in several different missions;
- the increased performance in specified applications. Due to the increased area covered by a fractionated/swarm system, unrestricted measurements are allowed and, consequently, a higher measurement accuracy is obtained;
- the increased reliability. The failure of one functional module does not compromise the mission as it can be substituted by another.

All the aforementioned advantages contribute to the core benefit of this technology: its increased potential for scalability. Mathieu et al. [29] studied the impact of fractionation of spacecraft through different mesures of scalability, including "cost ratio between the additional scaled-down payload module and the corresponding scaled-up traditional spacecraft" and "time necessary to get the increased performance level", and both parameters improved with fractionation.

The potential for increased survivability of these spacecraft compared with monolithic spacecraft has triggered research analysis on the possibility of converting current missions into fractionated spacecraft missions.

ESA's Report on Fractionated Satellites [30] details a comparative assessment between monolithic and fractionated spacecraft in current LEO, GEO, L1 and planetary missions. The report concludes that the most appropriate missions for the conversion are LEO and GEO missions, where the added propulsion is not significant. DARPA's F6 Project [31] aimed at demonstrating the feasibility and benefits of a satellite architecture wherein the functionality of a traditional monolithic spacecraft is delivered by a cluster of wirelessly-interconnected modules capable of sharing their resources and utilising resources found elsewhere in the cluster. However, on May 16, 2013, DARPA confirmed that the project was closed without a flight demonstration. ANDESITE (Adhoc Network Demonstration for Spatially Extended Satellite-based Inquiry and Other Team Endeavors) [32] is a 6U CubeSat with eight deployable picosatellites, which aims at flying a local network of magnetometers through the electrical currents that cause the Northern Lights.

Although no significant space demonstrations of fractionated spacecraft have been made, the robustness provided by swarm technology has contributed to the growing investment of the space sector in these systems.

Satellite Swarm Applications—Satellite swarm is a concept of distributed satellite, widely explored in the CubeSat context [33]. Satellite swarms have been proposed for multiple earth and deep-space exploration applications.

Regarding earth applications, swarm systems are mainly used for synthetic aperture radars applications such as Earth resource exploration, battlefield reconnaissance, natural disaster surveillance, etc [6, 7], as well as distributed sensor networks [34].

Regarding deep-space applications, swarm systems are mainly proposed for antenna arrays [35, 36] and exploration missions on asteroids, comets or planetary moons [24,26,37].

Several challenges are faced by spacecraft swarms in different applications such as available bandwidth for communications [5], required formation due to tracking or communication requirements [26], hardware limitations and processing power limitations [23] in specified exploration mission requirements, and competition with large state-of-the-art monolithic spacecraft [26].

Implementation of cooperation mechanism in swarms—Current research on swarm technology has begun to focus on the cooperation of agents towards finishing one complex mission [8]. Pang et al. [6] and Araguz et al. [9] developed scheduling algorithms to distribute tasks to agents and adapt to agent failure. Farrag et al. [10] proposed relay-communication agents to improve the communication of a swarm. Perez et al. [11] proposed a consensus model to detect and correct different faulty inputs in a satellite swarm. The consensus model uses redundancy of data to detect different types of errors. Izzo et al. [12] and Gazzi et al. [13] each proposed different dynamical models for the movement of agents in a swarm.

As the interest for increasingly autonomous multi-agent systems grows, swarm engineering research has leaned towards distributed ledger implementations as a viable decentralised approach [8]. Different distributed ledger technologies have been developed, namely, blockchain, Tangle, Hashgraph and Sidechain [38]. From the mentioned technologies, Hashgraph presents some security and centralised issues, whereas Tangle appears to be the most suitable for IoT applications. However, it also suffers of security and centralisation problems related to its implementation [38]. Therefore, blockchain seems to be the most suitable system for the swarm/fractionated spacecraft application.

Blockchain

A blockchain is a peer-to-peer network which can be described as a shared, immutable and distributed ledger. The ledger is stored and therefore visible by all participants, called nodes, and cannot be modified without a mutual agreement, which allows the establishment of a trusted transaction/communication between distrusted entities without the need for a centralised unit [21, 39]. Through a blockchain, any type of data, such as asset data or satellite commands, can be shared via transactions which are stored in blocks. Each block is composed of the hash (a cryptographic key) of the previous block, a nonce, a timestamp, a transaction, and its own hash, containing the previous hash. Consequently, the ledger cannot be modified : the modification of one added block will change its hash, resulting in the break of the chain and the loss of the next blocks.

Blockchain types—Various types of blockchain exist depending on the level of privatisation required [15, 19] :

- a public blockchain, which allows anyone to participate, join the network, read and exchange data. The most popular public blockchains are Bitcoin, with the Bitcoin cryptocurrency, and Ethereum, with the Ether cryptocurrency.
- a private blockchain, which can control who participates, joins the network, reads and exchanges data. These rules are defined at the network's creation.

Blockchain consensus mechanism—In order to obtain a distributed network and to eliminate all possibility of fraud [22], blockchain works with a consensus mechanism that allows the mutual agreement of all the nodes on the current state of the blockchain [40] and to add new blocks in the blockchain [41]. The most used protocol is Proof of Work (PoW), in which each node competes on a mathematical problem to add a new block [41]. However, PoW causes a huge waste of computing resources [17]. Other protocols exist, such as :

- Proof of Stake (PoS), which considers the voting 'weight' of each node in the network. It drastically reduces the resources usage and brings down transaction times [41] ;
- Proof of Authority (PoA), where only appointed trusted nodes can maintain the blockchain [41]. However, this makes the system less decentralised;
- Byzantine fault tolerance (BFT), a feature of a distributed network to reach consensus even when some of the nodes in the network, less than 33 $\%$, fail to respond or respond with incorrect information [41].

Smart Contracts—To secure and automate the data generating, storing, accessing and processing [19] in blockchain, smart contracts are deployed on the network. A smart contract is an irreversible program residing on the blockchain, which encodes any set of rules for access control [19] and/or predefined features of the blockchain.

Once deployed, a smart contract is identified by a contract address which is used to call its functions. Nodes can trigger the contracts by sending a transaction to the smart contract address with a call of these functions and the required arguments. The called functions will be executed (by all the nodes maintaining the blockchain) with the current state of the blockchain when this transaction is added in a block [39,41]. Additionally, transactions always include a payment for the execution. While reading functions are free, setting functions have a cost.

Oracles— In several applications of blockchain in space, the blockchain's interaction with external sources is done via transactions performed by users outside the blockchain, requesting specific data contained in the network. However, many applications built on blockchain need, themselves, to interact with other external systems in order to update data within the blockchain, enable a pending transaction which requires external information, etc. Due to the blockchain's inability to recover information not present in the network, a mechanism through which the blockchain is able to obtain data from external sources was developed.

A blockchain oracle is a mechanism that fetches data from the external world to include it in the isolated execution environment of a blockchain [42]. The information, which could range from sensor data to payment receipts or price fluctuations, is fed through smart contracts and triggers predefined actions. There are five different classifications of oracles [43]:

- Software Oracles, which handle information data that originates from online sources, like temperature, prices of commodities and goods, flight or train delays, etc.
- Hardware Oracles, which provide information directly from the physical world, for example, through movement or RFID sensors.
- Inbound Oracles, which provide data from the external world.
- Outbound Oracles, which provide smart contracts with the

ability to send data to the outside world.
• Consensus-based Oracles, which get their data from human consensus and prediction markets.

A diagram of a generalised oracle is presented in figure 1. The numbers indicate the order of the execution of functions.

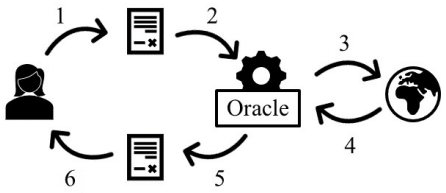

Figure 1. Generic Oracle Mechanism.

Typically, an oracle mechanism starts with a user's request to a smart contract (1). In the request, the user must specify the necessary data to trigger an oracle. Once triggered by a smart contract (2), the oracle will fetch the data (3, 4) and inject it in the blockchain through the execution of another smart contract (5). Finally, the latter smart contract will update the information requested (6). In the process of fetching data (3,4), oracles publish events on the blockchain, only accessible by external programs, which are in charge of detecting these event publications in order to run specific actions and call back functions of smart contracts when it is necessary.

Because blockchain oracles are generally off-chain components, the reliability properties of blockchains do not apply to them. Some kinds of data in the external world are inherently unable to be independently validated by multiple distributed parties, for example, because the data has restricted access, or is transient sensor data [42]. Nonetheless, the concept of consensus, typical to blockchain, can also be applied to the oracle mechanism in order to locally validate specific data. It is simply necessary that the mechanism is distributed i.e. there are several oracles fetching information for a given update.

Currently, there are several blockchain platforms with oracle mechanisms, such as Provable, TownCrier, Chainlink, Augur and Gnosis.

Blockchain Applications in the Space Sector—Despite having already been proposed for constellations and swarm systems, the use of blockchain has mainly focused on extending terrestrial blockchain networks with satellite nodes [20, 21] and increasing security in sat-com constellations [4, 15–19]. Note that Devi *et al* [44] proposed the use of blockchain to manage the interaction between subsystems in a monolithic spacecraft.

Most research proposes the use of blockchain to secure communication networks between satellites and between satellites and ground stations, as well as, to improve data sharing. Regarding the former application, a security increase is obtained through the implementation of access control [15, 16] or reputation-based [4, 17] protocols. Regarding the latter application, i.e. data sharing, network design solutions involve the use of a delay tolerance network [18] and a multi-granularity negotiation system to manage communication resources [8].

Torky et al. [14] presented several blockchain applications by using space digital tokens for orbits, satellites, spacecraft, orbital debris or asteroids. Here, the applications used blockchain to facilitate access to space and to space data.

The main disadvantage with blockchain in satellites is the need of significant amounts of power to execute protocols, and storage to store the blockchain [4,18,19]. Mital et al. [19] presented a way to manage capacity limitations of a satellite with Chain Trimming : as soon as the blockchain exceeds a specified size, data on the ledger is archived in a text file, sent to a ground station and a new chain is started.

The research path which has seen the largest industry implementations [16, 17, 20] has been the extension of terrestrial blockchain networks with satellite nodes in order to improve throughput in the blockchain [20, 21]. In particular, Ling et al. [20] proposed a consensus protocol in which GEO satellite constellations are used to generate and multicast random oracles to different nodes on a terrestrial blockchain network in order to increase the throughput of the network.

In another implementation, SpaceChain has developed an operating system deployed on satellites, providing a general blockchain application platform [16]. The project aims to bring an open-source decentralised software model to space [20].

3. Secured and Redundant Fractionated Spacecraft System

Context

In order to overcome the limitations of nano-satellites in space applications, a design of a fractionated spacecraft is proposed. The design facilitates the possibility for many different (educational or industrial) actors to contribute to a mission without having full confidence between themselves. Each actor or group of actors can develop one of the nano-satellites of the swarm, therefore implementing some of the subsystems of a monolithic spacecraft. The swarm is autonomous and managed by all actors. The security of the system is ensured by a trustless distributed ledger system, such as blockchain.

The aim of this work is :

• to present a particular type of fractionated spacecraft, composed of a swarm of nano-satellites, in order to reduce the cost of the mission per actor and increase the mission's performance when compared to equivalent monolithic spacecraft missions, as well as, facilitate scalability of the system and mission;

• to describe the integration of a blockchain in the swarm in order to increase the autonomy and the security of the mission, as well as, the reliability between the different entities involved.

The potential for scalability, characteristic to a fractionated system, comes at no extra implementation cost for the system, as the addition, from a new contributor, of an extra satellite to the swarm does not imply any modifications to the existing members. Furthermore, it allows for the maximisation and maintenance of redundancy through the increase of satellites. Finally, as previously stated, scalability increases the capacity of the system in specific applications such as antenna arrays, or distributed instruments [33].

In Figure 2, simplified representations of the proposed system alongside a typical monolithic spacecraft are presented. Redundancy of both systems is represented through various generic sensor icons. As can be seen, the proposed system is not necessarily symmetric, with satellites, as well as subsystems, which do not have the same number of sensors.

(a) Monolithic Satellite (b) Fractionated Spacecraft

Figure 2. Generic Representations of systems with Communication (yellow), Localisation (red) and Image (orange) Sensors.

Description of the system

In a first analysis of the concept, we assume that the fractionated spacecraft is composed of different subsystems. These subsystems request or provide some services to other subsystems. We consider that each subsystem is implemented on several satellites in order to improve the reliability and the availability of the global spacecraft.

In the context of a fractionated spacecraft where the number of nano-satellites can vary and where the reliability of each low-cost satellite is not fully ensured, our main objective is to defined a distributed system able to manage the global system.

Our proposal consists in deploying a blockchain on each satellite where a set of smart contracts manages the exchanges of services between the subsystems. For that, we propose to associate a smart contract to each subsystem. The different implementations of the corresponding subsystem play the role of oracle for the smart contract which aggregate their outputs in order to provide a validated output to the other subsystems. By extension, we will call this type of smart contract *Oracle Smart Contract*. Note that this can viewed as the implementation of redundancy, usually used for some critical embedded systems, including spacecrafts [45] in blockchains. We also introduce a second type of smart contract to manage the services between the smart contracts of the subsystems.

More formally, let us consider that the fractionated spacecraft is composed of NS satellites $S_1, \ldots, S_{NS}$. Each satellite contains some implementations of some of the ns subsystems $\mathtt{s}_1, \ldots, \mathtt{s}_{ns}$ of the fractionated spacecraft. The ns_j implementations of the subsystem $\mathtt{s}_j$ are denoted by $\mathtt{s}_{j,\sigma_j(1)}, \mathtt{s}_{j,\sigma_j(2)}, \ldots, \mathtt{s}_{j,\sigma_j(ns_j)}$, where $\sigma_j(i) \in \{1, \ldots, NS\}$ is the index of the satellite containing this implementation.

We assume that the oracle smart contract SC_j is associated to the subsystem $\mathtt{s}_j$. Since all the exchanges of services are done on the blockchain, any request r of the service j is performed by calling the smart contract $\mathrm{SC}_j(r)$ which transmits the requests to the implementations $\mathtt{s}_{j,\sigma_j(1)}, \ldots, \mathtt{s}_{j,\sigma_j(ns_j)}$. Each of them acts as an oracle and provides its output $\mathtt{s}_{j,\sigma_j(u)}(r)$ to $\mathrm{SC}_j(r)$ which aggregates them :

$$\mathrm{SC}_j(r) = \mathcal{A}(\mathtt{s}_{j,\sigma_j(n_1)}(r), \ldots, \mathtt{s}_{j,\sigma_j(ns_j)}(r)) \qquad (1)$$

where $\mathcal{A}$ is an aggregation function. According to the type of values to aggregate, some examples of aggregation are the choice of the most proposed value or the median value.

The second type of smart contract, called *Connector Smart Contract* is defined to manage interactions between the oracle smart contracts. Indeed, even if they can directly communicate together, some smart contracts need to gather the inputs of several oracle smart contracts in order to produce their result or to launch specific actions.

Functional principle

In the defined implementation, each satellite embeds a node of the blockchain and several subsystems. Additionally, each sensor is represented by a unique public key which serves as its identification in the blockchain interaction and can read and write data on the local blockchain node. Since, by the definition of blockchains, all nodes are synchronised, all collected data is accessible by all entities.

For example, when a request r is submitted to a subsystem $\mathtt{s}_j$:

1. a specific event containing the request r is published on a block of the blockchain, making it available to each blockchain node. Note that this step can be skipped if the request comes from a source external to the blockchain.
2. on each satellite S_i, if the corresponding subsystem sensor is implemented, as an external program, it detects the event, computes its result $\mathtt{s}_{j,i}(r)$ and calls a specific function of the smart contract to send the result to the blockchain (for example $addLocalOutput(\mathtt{s}_{j,i}(r))$.
3. after the synchronisation of the blockchain, all the outputs produced by the various implementations of the blockchain are available on each satellite.
4. then, the smart contract, implemented on each blockchain node, and thus on each satellite, runs the various calls to the function adding the outputs to its data (for example $addLocalOutput(\mathtt{s}_{j,i}(r))$. When it detects that enough data are available, it launches its aggregation function $\mathcal{A}(\mathtt{s}_{j,\sigma_j(n_1)}(r), \ldots, \mathtt{s}_{j,\sigma_j(ns_j)}(r))$ to produce the validated output of the subsystem $\mathrm{SC}_j(r)$.

Note that the implementation of a physical local network between satellites is therefore required but is out of the scope of this study.

It is interesting to observe that this scheme can be applied to several types of subsystems :

- a Symmetric Sensor Subsystem, responsible for receiving and processing data from sensors which receive approximately equal data.
- an Asymmetric Sensor Subsystem, responsible for receiving and processing data from sensors which receive differing data.
- the Communication Subsystem, responsible for receiving data from and transmitting data to the ground stations.

The Symmetric Sensor Subsystem case is the most simple. It can be applied, for example, to temperature sensors which measure roughly the same value.

The asymmetric sensor subsystem is quite different. It can be seen as a collaborative system between the sensors implemented on the various satellites. A good example is the *Absolute Position Sensor Subsystem*, responsible for receiving the absolute position of some of the satellites and,

through processing of data from all satellites, including relative positions, obtain the absolute position of the center of the system.

The last case is the Communication Subsystem. The antennas can be considered as sensors of the signal sent by ground stations. They are supposed to receive quite the same signal but according to the context, the smart contract can simply choose the message the most proposed by the antennas, or it can try to combine the signals received by the different antennas in order to improve, for example, the reliability.

Finally, the role of the *Connector Smart Contracts* is to manage the interactions between the *Oracle Smart Contracts*. Their functional principle is classical because they do not interact with entities external to the blockchain. Indeed, they are simply called by a *Oracle Smart Contract*, and according to the context, they call other *Oracle Smart Contracts*.

Risk Analysis

The benefits of using a decentralised architecture to increase a system's robustness are clear. However, as the complexity of the system increases, so does the susceptibility to failures and attacks, which, in turn, requires a more extensive risk analysis of the system.

As the blockchain is implemented on the satellites, but interacts with users, three main points of failure are detected : satellite nodes, ground station users and the environment. Similar risk analysis have been made for drone swarm missions [46, 47]. As in those missions, several high-level types of risks can be detected for each point of failure. In the following paragraphs, a list of such risks (similar to the list proposed by [46]) is presented and some counter-measures are proposed:

1. *Satellite Failure :* Satellite malfunctions can be caused by a collision with another spacecraft, another satellite or debris; abrasion due to harsh space environment conditions; software bugs; interference of charged particles with the system; etc. The two main consequences of this failure are:
- the satellite's inability to execute the required tasks of its subsystems. This must be detected by the system and notified to the ground station entities, which should then proceed to substitute the satellite. Additionally, as stated previously, the sheer number of satellites per subsystem must be such that the loss of a satellite is negligible to the subsystem's mission;
- the satellite's unpredictable behaviour towards the required tasks of its subsystems. This must be detected by the system and the satellite must be penalised or excluded from the system. The expulsion should be notified to the ground station entities.

2. *Malicious Satellite :* A satellite may be taken over by an adversary. In addition to the previously mentioned consequences, the satellite may :
- leak information shared by the network of satellites. A possible solution is the encryption of the system's data;
- launch DoS attacks by flooding the system with superfluous information. In such a case, a threshold of messages between processing batches can be implemented.

3. *Ground Station Failure :* A ground station user can experience failures, such as :
- the inability to connect to the system. Because such a case does not involve an interaction with the proposed system, its management is not discussed;
- the ground stations' unpredictable requests to the system. The behaviour should be detected by the system and the requests disregarded.

4. *Malicious Ground Station :* A ground station may be taken over by an adversary. The consequences are similar to the case of a malicious satellite and can be handled in the same manner.

5. *Environment Changes :* Space's harsh environment can have several impacts on the system. In addition to damages to the satellite's sensors, the communication between the system may be compromised. In such a case, the implementation of a retry loop, which repeatedly attempts to perform transactions until a successful output is achieved or until a maximum of attempts is met, may solve the problem.

4. USE CASE

The objective of this Section is to go deeper in the description of the system. For that, we will focus on a generic but practical scenario and then, we will show how to specify and implement the required functionalities on a blockchain.

Considered Scenario

We consider near-Earth orbital operations and make use of the blockchain to collaboratively measure data through different sensors, such as antennas, stellar sensors, GPS and measuring sensors. Please note that the fractionated spacecraft's mission is not limited to this orbit.

The chosen scenario is a sensing mission composed of three nano-satellites, all equipped with three different types of sensor (a temperature sensor, a communication sensor i.e an antenna, and an absolute position sensor). Consequently, each subsystem have the minimum number of three satellites, which allows testing of the detection of malicious satellites.

According to the context, a specific ground station can communicate with the satellites from the communication subsystem to request specific data. To obtain a better flexibility and resilience, we consider that all satellites belong to the communication subsystem, meaning that any authorised ground station can interact with any satellite. However, the ground stations consider the swarm as a unique entity and never communicate with only one satellite.

In the scenario, we assume that the ground station requests the value of a sensor. In a first step, we describe the general process without taking into account the specificity of the sensor. However, in later steps, we will analyse the case where the sensor is a simple temperature sensor (Symmetric Sensor Subsystem) and the case of absolute position sensor (Asymmetric Sensor Subsystem).

The proposed mission can recover sensitive space data, which the cooperating entities may not wish to disclose, therefore the network is defined as private. Thus, the access of satellites to the network is defined at the creation of the blockchain to assure the security of the system.

Additionally, due to limitations on the computing power of the satellite, PoW and PoS are not adapted for the proposed scenario, as a more energy efficient consensus protocol was necessary. In this sense, our choice is to implement a PoA network based on the Istanbul Byzantine Fault Tolerant 2.0 (IBFT 2.0) protocol.

To simulate the communication between the physical sensor and the established blockchain network, additional interface scripts are developed. The interface scripts are able to collect data from or send data to the associated oracle sensor, as well

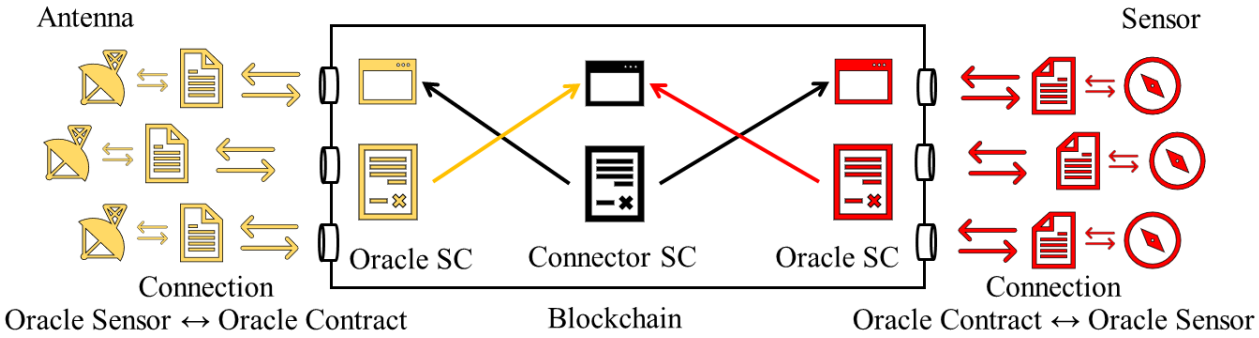

Figure 3. Software System Architecture.

as, listen to events emitted by and trigger a function from the associated Oracle Subsystem Smart Contract. They are implemented in a scripting language that supports event handling and interaction with a blockchain network (e.g Python).

In a simplified first approach to the system, each sensor is simulated through an individual text file containing the data they collected.

Proposed System

The implementation for the chosen scenario is developed, taking into consideration all the decisions made in the previous sections.

All the above defined subsystems are implemented using a general code structure so as to allow the work to be expanded and applicable in different application cases. To obtain a suitable structure for all subsystems the following points are taken into account:

- Independently of the subsystem, the developed code must allow for the interaction of both sensor systems and ground stations with the blockchain nodes through oracles. In both cases, this interaction should result in the processing of a given set of data and subsequent updates in the blockchain.
- Independently of the subsystem, the developed code must allow for a node from the fractionated system to interact with sensors through oracles. This interaction should result in the processing of a given set of data and subsequent updates in the blockchain.

To achieve the aforementioned requirements, the structure presented in figure 3 was developed.

In the diagram, smart contract files are represented by contract icons, interfaces for those contracts are represented as tabs and ports of the blockchain network (used to communicate with nodes) are depicted as cylinders. The arrows indicate the direction of the communication.

The three main components of the proposed implementation are therefore:

Connection Oracle Sensor ↔ Oracle Smart Contract—responsible for the interface between every sensor and the corresponding oracle contracts. Considering the direction from the blockchain to the sensors, these files listen to events emitted by oracle smart contracts and interact with the sensors to retrieve necessary data. In the opposite direction, sensors (such as antennas) can provide information to the files, which trigger them to call functions from oracle smart contracts.

Oracle Smart Contracts—responsible for the data processing of each subsystem. These files contain the necessary protocols to identify malfunctioning or malicious nodes and incoherent data, as well as manage such cases. Considering the direction from the blockchain to the sensors, these files' functions are called through their interface by the Connector Smart Contract to interact with the Connection files and request data from the sensors (or provide it in the case of antennas) through the emission of events. In the opposite direction, these files' functions are called by the Connection files, which transmit information through function parameters. After the data processing, this information is then transmitted to the Connector Smart Contract through the call of functions from this contract's interface.

Connector Smart Contract—responsible for the interface between different oracle contracts and, consequently, different sensor subsystems. It is a contract available to all nodes of the blockchain and accessible through an interface shared with the sensor's contracts. The call of a function from the Connector contract, by an Oracle contract (for example, the communication contract), will, subsequently, trigger the call of an additional function in another Oracle contract (for example, the symmetric sensor Oracle contract).

In order to better illustrate the flow of data in the system, a detailed description of the process following a request of data from a generic sensor (symmetric or asymmetric) by a ground station is presented in figure 4.

In the flowchart, function names are written in `verb` font, dash-dotted lines indicate that an event was emitted by one function and listened to by another and dashed lines indicate that multiple functions were called between two function calls. The numbers attached to each function indicate the order of execution of the functions. Functions regarding processing of data and voting protocols were not included in the flowchart to decrease cluttering of information. However, as stated previously, these functions belong to the oracle smart contracts and are called by the functions `askSensorValue` and `setLatestSensorValue`.

By analysing the various functions presented, it is possible to identify similarities between the two oracles contracts. Both contracts have a function called by a script file, responsible for processing received data and transmitting that data to the caller contract (`askSensorValue` and `setLatestSensorValue`). Similarly, both contracts have a function which is called by the caller contract and emits a specific event in order to transmit to or request data from the script files (`receiveSensorValue` and

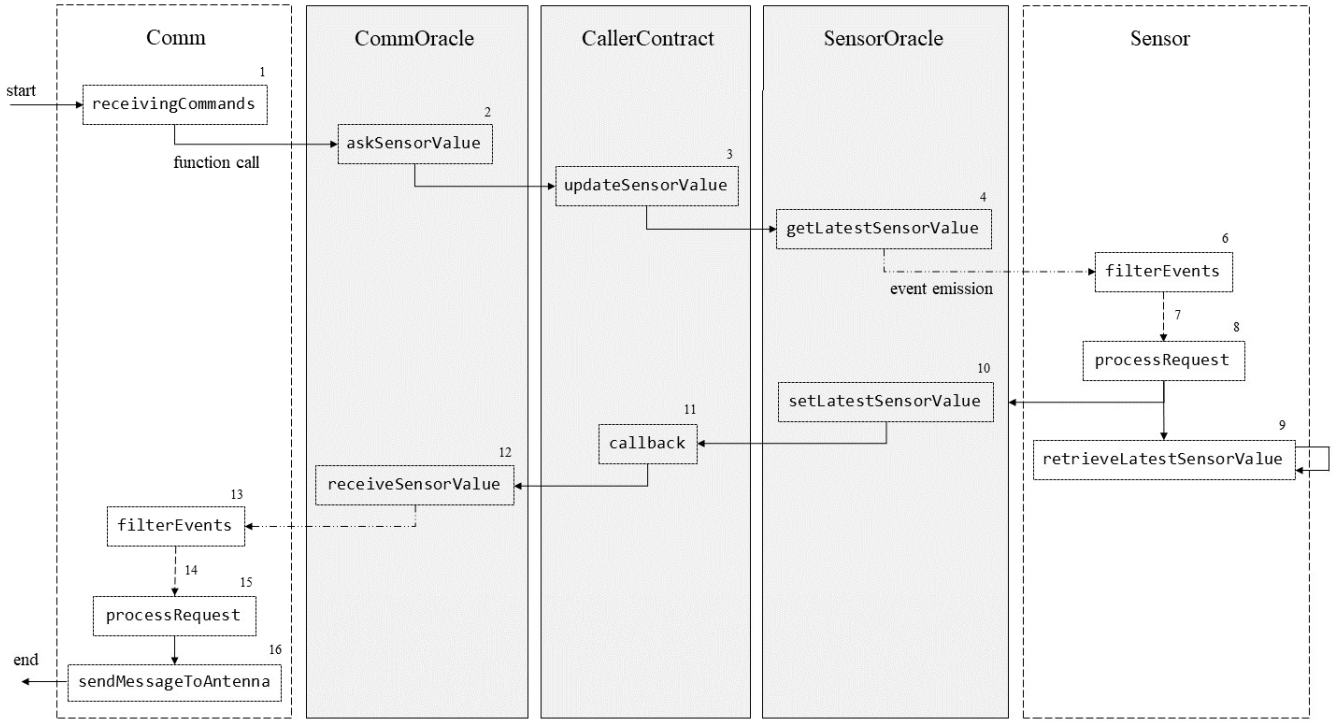

Figure 4. Flow of Data within the System.

`getLatestSensorValue`).

As for the script files, they both share functions responsible for listening to events of smart contracts (`filterEvents`, `processRequest` and `processQueue`, which was omitted for simplicity). Furthermore, both files have functions which receive and transmit data to the antenna and the sensor (`receivingCommands`, `sendMessageToAntenna` and `retrieveLatestSensorValue`).

Risk Management

A relevant characteristic of the defined architecture is the use of security blockchain features to protect the system from attacks.

Firstly, the blockchain is private i.e. every node of the blockchain is defined at the beginning of the mission and entry of new participants must be validated by the responsible entities.

Secondly, functions in the oracle smart contracts are only accessible to verified and validated users. As defined in this use case, each sensor of the system is represented by a **unique public key** and **address**. In the proposed implementation, both these values are stored in the script file of each sensor and are used to validate transactions to the blockchain, more specifically, to call functions from oracle contracts. Additionally, a list of approved sensor addresses is stored in each oracle smart contract. This list contains all approved sensors which collect data for that subsystem. The addition of elements to the list can only be done by members of the list. The removal of elements from the list can only be done by the elements themselves via an **auto-destruction** function or through the violation of an imposed reputation record defined by all participants.

Finally, the caller contract has no interaction with outside participants as it is only accessible through other smart contracts.

Alongside the aforementioned features chosen for the blockchain, several security protocols are implemented in order to manage the security risks discussed in section 3. The main security concerns are handled as follows

Validation of the received Data— In order to validate data from each subsystem, a minimum number of sensor responses is necessary. This number is stored in the variable **threshold**, defined in the beginning of the mission and can vary from subsystem to subsystem. After the threshold of responses is obtained, the set of responses is fed to an algorithm specific to each subsystem (aggregation function $\mathcal{A}$), which outputs a computed value.

In the communication and the symmetric sensor oracle, the computed result is the **mode** of the set of data received.

However, in the asymmetric sensor oracle, an additional computation is required before the aggregation function, i.e. calculating the mode. In the latter subsystem, the value retrieved from each sensor corresponds to the absolute position of the sensor, meaning it varies from sensor to sensor (hence the asymmetry of the subsystem). Therefore, it is necessary to calculate the absolute position of the system's center for each sensor before performing the mode. This calculation is possible, because, as stated in this use case, we assume that every satellite has access to the relative positions of all the satellites in the system. The absolute position of the system's center $\mathtt{a}_{j,i}$ obtained through the implementation of the absolute position subsystem $\mathtt{s}_j$, from the satellite S_i is

then given by

$$\mathtt{a}_{j,i} = \mathtt{s}_{j,i} + \frac{\sum_{k=1}^{NR} \mathtt{p}_{j,k,i}}{NR}, \quad (2)$$

where $\mathtt{s}_{j,i}$ is the absolute position obtained by satellite S_i, $\mathtt{p}_{j,k,i}$ is the relative position of satellite S_k to satellite S_i and NR is the number of relative positions the satellite S_i has access to.

Resistance to Denial-of-Service (DoS) attacks—The flooding of a subsystem with data is also managed through the variable **threshold** as it is not only the minimum but the maximum number of responses allowed to compute a sensor value.

In addition, the variable **pendingSensorRequests** is also used to stop sensors from transmitting repeated data to the subsystem. The latter variable is a mapping which associates the identification address of a sensor to a boolean value. If the boolean is "true", then the sensor has performed the request and the request has not yet been processed by the subsystem. While the boolean is "true", equivalent requests made from the same sensor are disregarded.

Resistance to malfunctioning or malicious sensors—In order to defend a subsystem against malfunctioning or malicious nodes, a **reputation record**, common to all validated sensors, is used.

Each sensor's reputation is updated when a new sensor value is computed. In the case of the symmetric and asymmetric sensor subsystems, in order to update the reputation record, the maximum variation allowed between the computed value and the retrieved value from a sensor must be fixed. If the sensor value is not contained within the maximum variation interval, a point is added to its reputation. This variation is stored in the variable **maxVariation**, defined at the beginning of the mission and specific to each subsystem.

Finally, the expulsion of under-performing sensors from a subsystem is ensured by the variable **repThreshold**, which is also defined at the beginning of a mission and specific to each subsystem. This variable corresponds to the maximum reputation record a sensor can have before being expelled from the subsystem. If a sensor surpasses this threshold, its address is removed from the validated sensors list.

Resistance to network shortages—In order to limit the data loss due to network shortages, the retry loop proposed in the risk analysis section is implemented in the script files.

If a transaction performed by a sensor oracle fails, the oracle waits for two seconds and re-executes the transaction. If, after a maximum number of tries (**MAXRETRIES**), the transaction fails, the oracle abandons the transaction and does not transmit its data to the blockchain.

Implementation choices

The chosen network is the Ethereum network because at the time of writing, it is the most established network which supports smart contracts. Additionally, this network's popularity can be attributed to its open source environment and the richness of its documentation [48].

Several proposals were made in order to deploy Ethereum clients on embedded systems [46] and even on the International Space Station [49]. They prove that with some modifications (out of the scope of this paper) and a correct choice of the parameters of the blockchain, it is realistic to plan to integrate light Ethereum clients on a nano-satellite.

Note that Istanbul Byzantine Fault Tolerant 2.0 (IBFT 2.0) protocol is adapted to the Ethereum network.

We selected Python as our Scripting language. Python supports event handling and interaction with a blockchain network, offers comprehensive analytic tools, is reliable and has an active developers community.

The proposed implementation's source code is available under an open-source GNU General Public License [50].

5. RESULTS

The final phase of research is the system's testing in an ethereum testing network.

Experimental Setup

The testing environment consists of an Ethereum network with four nodes running in the same machine. The smart contracts are deployed on the network and a system setup python file runs on the same host.

The setup file contains the address of the first member of the system and is responsible for adding all the validated addresses of each sensor to the respective oracle smart contract as well as define initial parameters of each of these contracts, namely, the **threshold** (= 3), the **maxVariation** (= 600) and the **repThreshold** (= 2).

Several text files are used to simulate the stored data from each sensor. A commands text file simulates the received requests from the antennas. The requested sensor values are also stored in a text file to simulate the response of the communication subsystem to the ground station.

Evaluation Metrics

In order to analyse the system's performance, the following parameters are obtained for each transaction in the blockchain:

- the **number of the mined block** where the transaction is saved. This parameter allows us to estimate the order of the transactions in the system;
- the **gas used** for the transaction. The gas given to miners after every transaction is strongly related to the processing power necessary to mine that transaction. As the proposed use case is composed of nano-satellites, whose processing power is limited, it is necessary to keep this metric as low as possible. Additionally, this measurement may be useful in similar use cases for public blockchains, where the gas price is not null and varies rapidly.
- the **status of the transaction** i.e. the boolean value which indicates if the transaction was successful or not. This parameter allows us to understand the behaviour of the global system and possible points of failure.
- the **account balance** of the sensor which performed that transaction. Similarly to the gas used, this cumulative parameter allows us to test the overall cost of tasks for each sensor in the system.

Simulation Results

Two phases of simulations were designed and performed to validate the system's implementation. In a first phase, each subsystem was tested individually to verify the specific security protocols as well as tasks performed by the subsystems. In the final phase, the whole system was tested.

Symmetric Sensor Subsystem—In order to test the symmetric sensor system, three sensor oracles were generated, each with a personal private key and port, simulating three satellites. Additionally, a supplementary python file was coded to interact with the caller contract and continuously demand data from the sensor system. The threshold of responses was set to three, meaning that all sensors were necessary to compute a data value. The reputation threshold was set as two.

In a first simulation, all three generated sensors were provided with identical sensor data. As expected, every request resulted in the return of a computed value and none of the satellites were expelled from the subsystem.

In a second simulation, all but one of the sensors were provided with identical sensor data. As expected, the first three requests resulted in the return of a computed value. After the third request, the malfunctioning satellite was expelled from the subsystem and the subsystem stopped generating sensor data. Additionally, the generated malfunctioning satellite shut down after detecting that it had been expelled from the system.

Regarding the evaluation metrics, every transaction had a status = 1 i.e. there were no errors with the transactions. Additionally, all the tasks were run sequentially i.e. the first sensor to obtain the data was also the first one to call the smart contracts and have a successful transaction. The average gas used on the most relevant transactions is presented in table 1.

Table 1. Interval range and Average of **gas used** in relevant transactions of the Symmetric Sensor Subsystem.

Function	Gas Used (Interval)	Gas Used (Average)
addOracle	[51592;66592]	56592
autoDestructOracle	[19376;23751]	22080
updateSensorValue	-	76421
setLatestSensorValue	[93796;160117]	117123

As expected the most expensive function in the oracle is the **setLatestSensorValue** function.

Asymmetric Sensor Subsystem—The asymmetric sensor system was simulated using the same scenario has the previous sensor. However, in this case, an additional sensor data file had to be provided for each satellite. These files contained the relative positions of every satellite in the system in relation to the given satellite.

Identical simulations were performed and similar results were obtained.

Regarding the evaluation metrics, every transaction had a status = 1 i.e. there were no errors with the transactions. Additionally, all the tasks were run sequentially i.e. the first sensor to obtain the data was also the first one to call the smart contracts and have a successful transaction. The average gas used on the most relevant transactions is presented in table 2.

Table 2. Interval range and Average of **gas used** in relevant transactions of the Asymmetric Sensor Subsystem.

Function	Gas Used (Interval)	Gas Used (Average)
addOracle	[22896;66614]	49328
autoDestructOracle	[19387;23773]	22099
updateSensorValue	-	76421
setLatestSensorValue	[79994;179947]	119357

As expected the most expensive function in the oracle is the **setLatestSensorValue** function.

Communication Subsystem—The communication subsystem was tested with two nodes by receiving commands from antennas. For this test, two oracles were generated, each with a personal private key and a port, simulating two satellites. Additionally, one text file was created with commands reading by satellites, simulating reception of same messages by each satellite. To simulate the forwarding of answers, the satellite which sent back the answer was randomly chosen by the program and wrote its answers in a new text file.

The first test was to send back to the antenna a received message : "great job". Each satellite received this message and used the blockchain to send it back in its answer file. As expected, the message was correctly written in the chosen answer file.

Then, the second test was under the same conditions with several messages to send back. As expected, all messages were correctly written in the two answer files.

The average gas used on the most relevant transactions is presented in table 3.

Table 3. Interval Range and Average **gas used** in relevant transactions of the Communication Subsystem.

Function	Gas Used (Interval)	Gas Used (Average)
addOracle	[78393; 108393]	93393
autoDestructOracle	[29162; 27853]	28507.5
sendMessageToAntenna	[58904; 73904]	60404

The first execution of **sendMessageToAntenna** function always uses more gas than next executions. Then, quantity of gas is the same for a same sent message and varies in proportion to the length of the message to send back. In function of the message length, current average price of the **sendMessageToAntenna** function varies.

Fractionated Spacecraft System—As for the complete system, the scenario detailed in the use case was simulated. Therefore, nine sensors, three for each subsystem (temperature, absolute position and communication), were generated. A different port of the blockchain network was associated to each satellite simulated. Consequently, for each subsystem, a different port was associated to every sensor. However, one sensor from each subsystem shared the same port (corresponding to the satellite they belonged to). A representation of the scenario is presented in figure 5.

In several simulations, the command text file was randomly filled with commands to request data from both sensor subsystems.

In a first phase of simulations, no malfunctioning or malicious

Figure 5. Concrete Scenario of the System.

nodes were generated. As expected, every request resulted in the corresponding response text file being filled in with the labelled sensor values.

In a second phase of simulations, with a malfunctioning node in each sensor subsystems, the expected results were obtained. The first three requests for each subsystem resulted in the return of a computed value. After the third request on each subsystem, the malfunctioning satellites were expelled from the subsystem and the subsystems stopped generating sensor data. Additionally, the generated malfunctioning satellites shut down after detecting that they had been expelled from the system.

Regarding the evaluation metrics, every transaction had a status = 1 i.e. there were no errors with the transactions. Additionally, all the tasks were run sequentially i.e. the first sensor to obtain the data was also the first one to call the smart contracts and have a successful transaction. The average gas used on the most relevant transactions is presented in table 4.

The most expensive function in the network is the **setLatestSensorValue** function for the asymmetric sensor system. The general trend of the system simulations was an increase in gas spent in each function. This result is coherent, seeing as, when testing the complete system, the functions from each oracle contract trigger additional functions from other oracle contracts.

The simulation results presented in this section validate the proposed proof-of-concept. It is possible to confirm that the developed blockchain is resilient to malicious nodes, node failure and incorrect sensor data.

6. Conclusion

The characteristics of decentralisation, difficult data tampering and traceability make blockchain technology an excellent network solution for the management paradigm of multi-satellite systems. In the present paper, a novel application for the blockchain network in a collaborative multi-entity fractionated spacecraft system is presented. The system is composed of functionally different satellites which perform tasks pertaining to different subsystems of an equivalent monolithic spacecraft. Transparency in the system is ensured by the network, allowing for missions between entirely trustless parties. This new framework allows for cooperation between sensors of different types (with different tasks) through the use of communicating blockchain oracles and facilitates scalability of the system. The proposed system is resilient to malicious nodes, node failure and incorrect sensor data.

Regarding the extension of the proposed architecture, a possible modification could be the implementation of entry deposits in ether for new sensors. The current reputation system could be strengthened by including penalties on the deposits every time a sensor provides poor data values. This protocol would consequently encourage sensors to perform tasks adequately.

A following step of the presented research is the implementation of the developed network in an embedded board with similar processing capabilities and memory storage to on-board computers used in nano-satellites. Successful simulation results will confirm that the defined implementation is valid and adequate for the proposed application, more specifically, the proposed hardware. In such a case, considering the results from previous research regarding resilience of swarm systems, it should also be possible to show that the proposed system is more resilient to node failure than a monolithic spacecraft to subsystem failure.

Outside of the presented use case, there are various applications for the proposed architecture of communicating oracles. The use of blockchain networks and oracles as already been proposed for sensor systems in IoT devices, to manage access control and data storage [51]. The autonomous multi-directional architecture of the developed network can similarly be adapted to IoT applications which require not only data validation, but also communication between devices. An example application is smart cars' management, which require continuous updating of sensor information and transmission of sensor data to external devices, namely, steering wheels and breaks of cars. In addition to the management of a single car, the management of a network of autonomous vehicles could also be implemented with the developed network, as long as communication between vehicles is guaranteed.

In conclusion, the use of blockchain networks in multi-satellite fractionated systems can bring several advantages

Table 4. Interval Range and Average **gas used** in relevant transactions of the Fractionated Spacecraft System.

Subsystem	Function	Gas Used (Interval)	Gas Used (Average)
SymmetricSensorOracle	addOracle	[51592;66592]	56592
	autoDestructOracle	[19376;23751]	22293
	setLatestSensorValue	[93808;149782]	117466
AsymmetricSensorOracle	addOracle	[51614;66614]	17205
	autoDestructOracle	[19387;23773]	22311
	setLatestSensorValue	[98834;167236]	126511
CommunicationOracle	addOracle	[78393;108393]	88393
	autoDestructOracle	[29162;30941]	30052
	askLatestSensorValue	[96387;114024]	107266

to collaborative missions in terms of trust, resilience and security.

REFERENCES

[1] J. Thangavelautham, M. Herreras-Martinez, A. Warren, A. Chandra, and E. Asphaug, "The suncube femtosat platform: A pathway to low-cost interplanetary exploration," in 6th Interplanetary CubeSat Workshop, Oxford, England, 2016.

[2] H. Heidt, J. Puig-Suari, A. Moore, S. Nakasuka, and R. Twiggs, "Cubesat: A new generation of picosatellite for education and industry low-cost space experimentation," 2000.

[3] C. Cappelletti and D. Robson, "Cubesat missions and applications," in Cubesat Handbook. Elsevier, 2021, pp. 53–65.

[4] L. Clark, Y.-C. Tung, M. Clark, and L. Zapanta, "A blockchain-based reputation system for small satellite relay networks," in 2020 IEEE Aerospace Conference. IEEE, 2020, pp. 1–8.

[5] S. Engelen, E. K. Gill, and C. J. Verhoeven, "Systems engineering challenges for satellite swarms," in 2011 Aerospace Conference, 2011, pp. 1–8.

[6] C. K. Pang, A. Kumar, C. H. Goh, and C. V. Le, "Nano-satellite swarm for sar applications: Design and robust scheduling," IEEE Transactions on Aerospace and Electronic Systems, vol. 51, no. 2, pp. 853–865, 2015.

[7] F. Y. Hadaegh, S.-J. Chung, and H. M. Manohara, "On development of 100-gram-class spacecraft for swarm applications," IEEE Systems Journal, vol. 10, no. 2, pp. 673–684, 2014.

[8] X. Deng, Y. Dong, and S. Xie, "Multi-granularity mission negotiation for a decentralized remote sensing satellite cluster," Remote Sensing, vol. 12, no. 21, p. 3595, 2020.

[9] C. Araguz, M. Closa, E. Bou-Balust, and E. Alarcon, "A design-oriented characterization framework for decentralized, distributed, autonomous systems: the nano-satellite swarm case," in 2019 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2019, pp. 1–5.

[10] A. Farrag, S. Othman, T. Mahmoud, and A. Y. EL-Raffiei, "Satellite swarm survey and new conceptual design for earth observation applications," The Egyptian Journal of Remote Sensing and Space Science, 2019.

[11] I. Perez, A. Goodloe, and W. Edmonson, "Fault-tolerant swarms," in 2019 IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT), 2019, pp. 47–54.

[12] D. Izzo and L. Pettazzi, "Autonomous and distributed motion planning for satellite swarm," Journal of Guidance, Control, and Dynamics, vol. 30, no. 2, pp. 449–459, 2007.

[13] V. Gazi and K. Passino, "Stability analysis of swarms," IEEE Transactions on Automatic Control, vol. 48, no. 4, pp. 692–697, 2003.

[14] M. Torky, T. Gaber, and A. E. Hassanien, "Blockchain in space industry: Challenges and solutions," arXiv preprint arXiv:2002.12878, 2020.

[15] S. Wei, S. Li, P. Liu, and M. Liu, "Bavp: Blockchain-based access verification protocol in leo constellation using ibe keys," Security and Communication Networks, vol. 2018, 2018.

[16] S. Cao, S. Dang, Y. Zhang, W. Wang, and N. Cheng, "A blockchain-based access control and intrusion detection framework for satellite communication systems," Computer Communications, 2021.

[17] S. Cheng, Y. Gao, X. Li, Y. Du, Y. Du, and S. Hu, "Blockchain application in space information network security," in International Conference on Space Information Network. Springer, 2018, pp. 3–9.

[18] M. Feng and H. Xu, "Msnet-blockchain: A new framework for securing mobile satellite communication network," in 2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON). IEEE, 2019, pp. 1–9.

[19] R. Mital, J. de La Beaujardiere, R. Mital, M. Cole, and C. Norton, "Blockchain application within a multi-sensor satellite architecture," in The Adv. Maui Opt. Space Surveillance Technol. Conf, 2018.

[20] X. Ling, Z. Gao, Y. Le, L. You, J. Wang, Z. Ding, and X. Gao, "Satellite-aided consensus protocol for scalable blockchains," Sensors, vol. 20, no. 19, p. 5616, 2020.

[21] H. Wei, W. Feng, C. Zhang, Y. Chen, Y. Fang, and N. Ge, "Creating efficient blockchains for the internet of things by coordinated satellite-terrestrial networks," IEEE Wireless Communications, vol. 27, no. 3, pp. 104–110, 2020.

[22] W. Li, Z. Su, R. Li, K. Zhang, and Y. Wang, "Blockchain-based data security for artificial intelligence applications in 6g networks," IEEE Network, vol. 34, no. 6, pp. 31–37, 2020.

[23] C. J. Verhoeven, M. J. Bentum, G. Monna, J. Rotteveel, and J. Guo, "On the origin of satellite swarms," Acta Astronautica, vol. 68, no. 7-8, pp. 1392–1395, 2011.

[24] R. Sterritt, G. Wilkie, C. Saunders, M. Doran, C. Gama, G. Hawe, and L. McGuigan, "Inspiration for space 2.0 from autonomous nanotechnology swarms concept missions towards autonomic robotic craft," Journal of the British Interplanetary Society, vol. 73, no. 11, p. 397, 2020.

[25] A. Golkar, "Distributed cubesat mission concepts," in Cubesat Handbook. Elsevier, 2021, pp. 123–133.

[26] R. t. Nallapu and J. Thangavelautham, "Attitude control of spacecraft swarms for visual mapping of planetary bodies," in 2019 IEEE Aerospace Conference, 2019, pp. 1–16.

[27] C. Mathieu and A. Weigel, "Assessing the flexibility provided by fractionated spacecraft," in Space 2005, 2005, p. 6700.

[28] L. Chi, F. Sun, Z. Liu, and C. Lin, "Overview of fractionated spacecraft technology," in 2020 International Conference on Computer Engineering and Application (ICCEA). IEEE, 2020, pp. 689–692.

[29] C. Mathieu and A. Weigel, "Assessing the fractionated spacecraft concept," in Space 2006, 2006, p. 7212.

[30] B. G. J. F. Dufour, C. Cougnet, "Fractionated satellites," 2010.

[31] "System F6 (Archived)," https://www.darpa.mil/program/system-f6, accessed in 2021-06-20.

[32] "ANDESITE 6U CubeSat Auroral Plasma

Science Mission of Boston University," https://directory.eoportal.org/web/eoportal/satellite-missions/content/-/article/andesite, accessed in 2021-06-20.

[33] C. Cappelletti and D. Robson, "From mission design to operations," in Cubesat Handbook. Elsevier, 2021, pp. 53–65.

[34] T. J. Sabaka, L. Tøffner-Clausen, N. Olsen, and C. C. Finlay, "A comprehensive model of earth's magnetic field determined from 4 years of swarm satellite observations," Earth, Planets and Space, vol. 70, no. 1, pp. 1–26, 2018.

[35] A. Budianu, R. Rajan, S. Engelen, A. Meijerink, C. Verhoeven, and M. Bentum, "Olfar: Adaptive topology for satellite swarms," in International Astronautical Congress. Citeseer, 2011, pp. 3–7.

[36] "SULFRO: a Swarm of Nano-/Micro-Satellite at SE L2 for Space Ultra-Low Frequency Radio Observatory," https://digitalcommons.usu.edu/smallsat/2014/AdvTechI/8/, accessed in 2021-05-09.

[37] P. D'Arrigo and S. Santandrea, "Apies: A mission for the exploration of the main asteroid belt using a swarm of microsatellites," Acta Astronautica, vol. 59, no. 8-11, pp. 689–699, 2006.

[38] N. El Ioini and C. Pahl, "A review of distributed ledger technologies," in OTM Confederated International Conferences" On the Move to Meaningful Internet Systems". Springer, 2018, pp. 277–288.

[39] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, 2016, pp. 254–269.

[40] S. Falcone, J. Zhang, A. Cameron, and A. Abdel-Rahman, "Blockchain design for an embedded system," Ledger, 2019.

[41] V. Strobel, E. Castelló Ferrer, and M. Dorigo, "Managing byzantine robots via blockchain technology in a swarm robotics collective decision making scenario," 2018.

[42] S. K. Lo, X. Xu, M. Staples, and L. Yao, "Reliability analysis for blockchain oracles," Computers & Electrical Engineering, vol. 83, p. 106582, 2020.

[43] "Blockchain Oracles," https://blockchainhub.net/blockchain-oracles/, accessed in 2021-06-20.

[44] M. S. Devi, R. Suguna, and P. Abhinaya, "Integration of blockchain and iot in satellite monitoring process," in 2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT). IEEE, 2019, pp. 1–6.

[45] J. R. Sklaroff, "Redundancy management technique for space shuttle computers," IBM Journal of Research and Development, vol. 20, no. 1, pp. 20–28, 1976.

[46] C. C. Mário G. Santos De Campos, Caroline P. C. Chanel and J. Lacan, "A mission-level resilient blockchain-based robotic system," IEEE T-RO Special Issue on Resilience in Networked Robotic Systems, 2020.

[47] S. R. Pokhrel, "Blockchain brings trust to collaborative drones and leo satellites: An intelligent decentralized learning in the space," IEEE Sensors Journal, 2021.

[48] G. Wood et al., "Ethereum: A secure decentralised generalised transaction ledger," Ethereum project yellow paper, vol. 151, no. 2014, pp. 1–32, 2014.

[49] "SpaceX to Send First Ethereum Node to ISS: Here's How Elon Musk's Space Agency and SpaceChain Will Keep ETH Safe," https://www.techtimes.com/articles/260911/20210601/spacex-send-first-ethereum-node-iss-heres-elon-musks-space.htm, accessed in 2021-10-06.

[50] M. Alves, J. Veirier d'Aiguebonne, T. Gateau, and J. Lacan, "Fractionated-Spacecraft-Blockchain," 10 2021. [Online]. Available: https://github.com/Mariana-Andrade-Alves/Fractionated-Spacecraft-Blockchain

[51] H. Al Breiki, L. Al Qassem, K. Salah, M. H. U. Rehman, and D. Sevtinovic, "Decentralized access control for iot data using blockchain and trusted oracles," in 2019 IEEE International Conference on Industrial Internet (ICII). IEEE, 2019, pp. 248–257.