A Decentralised Location-Based Reputation Management System in the IoT using Blockchain

Ponlawat Weerapanpisit[a], Sergio Trilles[a*], Joaqu´ın Huerta[a], Marco Painho[b]

[a]Universitat Jaume I, Castellón, Spain

[b]Universidade Nova de Lisboa, Portugal

*e-mail: strilles@uji.es

This is the accepted version of the article published by *IEEE journal*

Weerapanpisit, P., Trilles, S., Huerta, J., & Painho, M. (2022). A Decentralised Location-Based Reputation Management System in the IoT using Blockchain. IEEE Internet of Things Journal. [Advanced online publication on 31 January 2022]. https://doi.org/10.1109/JIOT.2022.3147478

Funding

This study was supported by the TRUST4IoE project of the Programa Estatal de Proyectos de I+D de Generaci´on de Conocimiento of the Spanish government (grant number PID2019-104065GA-I00). Sergio Trilles has been funded by the postdoctoral Juan de la Cierva fellowship programme of the Spanish Ministry for Science and Innovation (IJC2018-035017-I).

A Decentralised Location-Based Reputation Management System in the IoT using Blockchain

Ponlawat Weerapanpisit, Sergio Trilles, Joaquín Huerta, Marco Painho

***Abstract*—The Internet of Things allows an object to connect to the Internet and observe or interact with a physical phenomenon. The communication technologies allow one IoT device to discover and communicate with another in order to exchange services, in a similar way to what humans do in their social networks. Knowing the reputation of another device is important to consider whether it is trustworthy before establishing a new connection and thus avoid possible unexpected behaviours as a consequence. Trustworthiness, as a property of a device, can be affected by different factors including its geographical location. Hence, this research work proposes an architecture to manage reputation values of end devices in an IoT system based on the area where they are located. A cloud-fog-edge architecture is proposed, where the fog layer uses the Blockchain technology to keep the reputation management system consistent and fault-tolerant across different nodes. The location-based part of the system was done by storing geographical areas in Smart Contracts (coined as Geospatial Smart Contracts) and making the reputation values subject to different regions depending on the geographical location of the device. To reduce the complexity of the spatial computation, the geographical data are geocoded by either one of two different spatial indexing techniques. This work also introduced two different structures for storing geocoded areas based on either cell-list or tree-structure. Finally, three experiments to test the proposed architecture are presented, to deploy the architecture in IoT devices, and to compare the two geocoding techniques in Smart Contracts.**

***Index Terms*—Internet of Things, Blockchain, Location-Based Trust, Spatial Indexing**

I. Introduction

THE INTERNET OF THINGS (IoT) has recently begun to play an important role in connecting physical objects to the Internet network. The development of hardware capabilities and communication technologies allows objects (or things) in our daily lives that are equipped with sensors or actuators to connect to the Internet (1). This process turns them into smart objects that can interact with users or with other devices in order to observe a physical phenomenon and perform an interaction (2).

In our society, there might be a number of service providers offering services that we need. When a person wants to interact or consume the service from another, it is necessary to know how trustworthy the providers are, before deciding to choose and start an interaction (3). This trustworthiness can be known based on experienced users' recommendations, as well as by evaluating service quality. This is necessary to ensure that the service will satisfy users and will not cause consequent failures.

In the same way, IoT devices can also communicate with each other in order to consume or provide services. Trustworthiness management is a component that is relevant to relationship management in human social networks. To ensure that the services provided are reliable and do not lead to consequent failures, it is important to consume the services from a trusted provider. Despite the fact that trust is a subjective concept and differs in each individual agent, it can be influenced by a quantitative value such as a reputation index, which is a social quantity property of an agent (4). In consequence, it is necessary for an IoT system to have an architecture that allows management of *reputation indices* for devices in the system. The values can be taken by a device consumer to decide whether they would trust the provider before establishing a new connection.

Devices in an IoT system are usually distributed over geographical space and are able to move across different areas. The location of the devices can be a factor that affects their trustworthiness (5). In other words, if two IoT devices have already interacted successfully in an area, a new interaction in the same area is more likely to be satisfactory. In this way, the spatial context of a device is crucial to set reputation values. In this work, we take the geospatial context to build a reputation management system architecture to manage IoT relationships, where each defined area (or geofence) will have its own reputation indices for each device. A possible application of this feature is to support automatic transport payments (taxis, buses,...) inside an area, where they establish connections between users' mobile phones to perform payments. These connections are produced in an area and based on previous knowledge; we can automatically accept the connections (payments). Another scenario could be the interaction between an irrigation management system and a nearby weather station. Where the management system would deactivate irrigation when the season indicated rain and the reputation within the area was reliable.

Another related technology is cloud-fog-edge architecture, which is a hierarchical arrangement in the IoT. It divides the system into three layers, namely, *cloud*, *fog* and *edge* (6). While the traditional cloud and edge layers were designed for heavy processing and low-level computation, respectively, the fog layer was proposed at a later stage by Cisco as an intermediate layer between the other two (7). The capacity of

P. Weerapanpisit, S. Trilles and J. Huerta are at Universitat Jaume I, Castellón, Spain (e-mail: {al394260,strilles,huerta}@uji.es) and M. Painho is at Universidade Nova de Lisboa, Portugal (e-mail: painho@novaims.unl.pt)

Manuscript received December 07, 2021; revised X Y, Z.
Digital Object Identifier 10.1109/JIOT.2020.XXXXXXX

fog hardware is generally lower than that in the cloud, but sufficiently greater than edge devices to allow it to perform more complex calculations. Hence, not all unnecessary computations have to be loaded into just the cloud layer. Moreover, fog devices are installed as a solution between the cloud and the edge layers. They are therefore generally distributed geographically, which is ideal to serve the purposes of the location-based reputation management system.

The last related concept is decentralisation. Due to the existence of a fog layer, the intermediate devices are now distributed and not all computations need to be performed in the cloud layer. Blockchain is a technology that allows data to be stored in a distributed way. All nodes in a Blockchain network will share and possess the same data, which are formed in blocks that are chronologically linked to each other like a chain. This chain is distributed over the network, which makes the system transparent and fault-tolerant (8). The Blockchain technology relies on hashing and consensus algorithms to confirm that all nodes in the network have the same valid data, and it is not easy to tamper with the chain. The implementation of the Blockchain is well-known in the field of cryptocurrencies, such as Bitcoin and Ethereum. But in Ethereum, besides storing the money transactions that take place, it also allows Smart Contract (an executable program) to be stored and executed from the Blockchain network.

Thus, this research work proposes the adoption of the Smart Contract application in the fog layer to create a decentralised application that is able to manage the reputation indices of an IoT system. However, even though decentralised applications can execute any program as a real computer does, the characteristics of the Blockchain limit the capacity to perform complex calculations in Smart Contracts. Geometric spatial objects and their calculation, which are related to complex algorithms and complicated data structures, could have an expensive computational cost (9). For this reason, we use spatial indexing techniques in the implementation to study and challenge the possibility of manipulating spatial data in the Blockchain. Two hierarchical spatial indexing techniques are studied in this work. The first one is Geohash, which is an indexing technique invented by Gustavo Niemeyer (10), and the second is S2, which was invented by Google engineers[1]. Both techniques share the same hierarchical characteristics despite the different algorithms behind them.

There are several related works that address an IoT system architecture to manage reputation values and trust. The authors in (11) proposed an architecture with five layers: reputation management, organisation, Soft Defined Network (SDN) control, node and object layers. Users request an operation of the IoT device through the organisation layer, which is midway between the reputation management centre and the end nodes. The organisation layer decides whether the requested node is trustworthy before executing the action. The authors in (12) proposed an interesting scenario of moving IoT devices and an architecture to manage their trust values. The work is based on a scenario involving the sharing of air quality data, the trustworthiness of which is evaluated by the experience of one user regarding another user in a different area. The consumer chooses the most trustworthy data provider and decides whether the air quality in the target area is satisfactory, so that they can move into the area. The work is not based on the reputation value, which is a common expectation value towards one agent in the system, but instead on subjective trust, which is a one-to-one relationship between devices.

There are also studies that attempted to decentralise the trust management system as well. The work conducted by (13) proposed an architecture of a decentralised reputation management system in an Ethereum network by using Smart Contract. In this work, end devices are in charge of evaluating the fog devices and storing their reputation values in Smart Contract. The reputation value in this work is subjected to fog devices, not to edge devices. Additionally, (14) has proposed a cloud-fog-edge-based architecture to manage trust values of devices in the system. The Blockchain network is implemented in the fog layer. Then, an end device generates the reputation information of another device in a transaction, and signs it before submitting it to the Blockchain network. However, the reputation value itself is the same even if the device has moved to a different region.

Regarding the geocoding techniques, (9) compared different techniques of geocoding between raw geographical objects (coordinates), the Z-Order space-filling curve (Geohash) and the Hilbert space-filling curve, in terms of computation, efficiency and utility. The study showed that geocoding using the Hilbert curve performed better in most of the aspects. (15) used Geohash and S2 to fit a desired region. The resulting cells were used as a geofence, stored in a Smart Contract. The work demonstrated the feasibility of handling spatial data in Smart Contracts. They finally concluded that in their work S2 performs better than Geohash.

Based on the aforementioned related works and the geospatial concept, we can define the following benefits and limitations of creating a new architecture to manage reputation. As benefits, establishing a three-layer architecture based on edge-fog-cloud can be helpful to achieve a better use of computing capacity. Another benefit demonstrated in the literature is the use of Blockchain to save the reputation levels of edge devices so that they can be shared in a reliable way. The above-mentioned related works have a series of limitations. The first is that edge devices are not carrying out their actions by themselves in an autonomous way and need the cloud layer to decide what action to perform. The second limitation is present in some studies where the management system is centralised in the cloud, as all the trust values of each device relationship are stored there, and therefore the calculation and storage of the values are dependent on the cloud layer. Another point to be improved is in the studies where a decentralised architecture is used through Blockchain. These studies do not take advantage of the geospatial component that the fog offers, because it is a layer distributed by geographical areas. Thus, no previous work explored the geospatial context using geocoding

[1]https://github.com/google/s2geometry

techniques to manage the reputation values.

The main objective of this work is to propose a decentralised IoT architecture that allows the location-based management of device services and their reputation values using Blockchain. To accomplish this main objective, 1) this work will propose an IoT architecture based on the cloud-fog-edge structure, and decentralise the management system in the fog layer using Smart Contracts; 2) secondly, the location-based management part of the system will be performed by storing geographical areas in Smart Contracts (we coined the term Geospatial Smart Contracts) and making the reputation values subject to different regions depending on the geographical location of the device; 3) as a way to reduce the complexity of the spatial computation in Smart Contracts, this paper will study and compare two spatial indexing techniques, Geohash and S2, which are used to represent the geographical data in the system and are two different structures for storing geocoded areas based on either cell-list or tree-structure; 4) finally, the architecture will be simulated by implementing and deploying the system in real IoT devices.

The rest of this article is structured as follows. Section II presents the background to position the current work. Section III explains the proposed architecture adding the tools and technologies used. After that, Section VII shows the experiment results of the architecture. This is followed by the last section (Section VIII), which concludes the work and summarises the experiment results, as well as suggesting steps that can be taken in future work.

II. Background

A. Internet of Things

The term Internet of Things refers to the combination of network (Internet) and physical objects (things). It was first coined in 1999 by Kevin Ashton in his work on using Radio Frequency IDentification (RFID) in a supply chain system (16). IoT devices rely on wireless communication technologies to connect them to the network. Such technologies that allow the development of IoT include Bluetooth, RFID, Wi-Fi or GSM. As the technologies in wireless communication are continuously being developed and advanced, it allows the IoT community to expand and grow significantly (17).

Social Internet of Things (SIoT) is an integrated concept involving IoT and human social networking. The idea is that the things in an IoT system can discover the other devices that provide the necessary services and then establish a relationship and communicate with them (18). As humans do in their social life, when a person wants to know someone or to use a business service, they must know how reliable it is. Also in computer systems, when a device wants to use a service from the other devices, before establishing a connection with them, it should know whether the source is trustworthy in order to avoid possible problems or failures. Nevertheless, the concept of trust is very subjective to each individual. The authors in (4) proposed that trust is a subjective expectation that one agent has towards another. It is used to form expectations about their future behaviours based on past events. In the work by (19), they divided the obtainment of trust in a computer system into two categories: policy-based trust and reputation-based trust. Policy-based trust is centralised and the decision criteria are based on a third party. The second one is based on reputation, which is a quantitative property derived from the observed actions or behaviours carried out by an agent in the past. Hence, the non-centralised characteristic of reputation-based trust allows an individual to subjectively decide on its trustworthiness.

As the IoT is growing and its related technologies are moving forward, an IoT system could expand to a larger number of devices and connected sensors. This raises consequent issues such as heavy processing and big data storage in the cloud layer or exceeding bandwidth in the network. To tackle the problem, the Cisco company proposed a solution by adding an intermediate layer between the cloud and end device (edge) layers, called the fog layer (7). In a general cloud-fog-edge architecture, the edge layer is the bottom most layer where the end devices are. The devices in this layer are the simplest and have less computational ability, and they are connected to sensors or actuators in order to observe physical phenomena or to take part in an interaction. There is generally a greater number of end devices and they should not perform any complex computation because they have limitations regarding hardware specification, memory and power consumption. Secondly, the fog layer is the intermediate layer. The devices in this layer have more computational ability and can handle preliminary data processing as well as store sensory data before forwarding them to the cloud layer. In the opposite direction, the fog layer can also be a middle party that passes commands or messages from the cloud layer to the edge layer. Because the fog layer can also be distributed geographically, it can organise the edge devices in the area under its responsibility. The last one is the cloud layer, which is the topmost layer in the architecture and is in charge of processing final data, managing the whole system and storing the sensory data. A cloud device is supposed to be physically static and located in a data centre or a dedicated place. The device itself can be either a dedicated server where the organisation administrator is responsible for administration and maintenance or it can be a cloud service in the form of a platform-as-a-service (PaaS) or a software-as-a-service (SaaS) which is provided by an external cloud service provider such as Microsoft Azure, Amazon Web Service (AWS) or Google Cloud.

B. Blockchain

Blockchain is a technology to store computer data in a distributed and decentralised way (20). A Blockchain network consists of a number of blocks. In a block there are transactions to store the data in the network. Blocks and transactions are uniquely identified by using a cryptography hashing algorithm.The identifier hashes of the blocks are used to link each other in a chronological linear sequence like a chain, which is the reason behind its name. Blockchain was originally mentioned by (21). It proposed an electronic cash system that can store the transactions of ledgers in a decentralised way by using peer-to-peer communication. A

Blockchain network has a number of blocks that are linked to each other in a chronological sequence. As Blockchain was originally developed for storing money movements in an electronic currency called Bitcoin, the data in each block are a set of money transactions. A Blockchain node collects transactions from its clients and packs them into one block, then, it pushes the block into the end of the chain. The node that has recently pushed the block will then broadcast the change to the other nodes in the same network to update the data. Finally, the other nodes verify the change before updating their own chain.

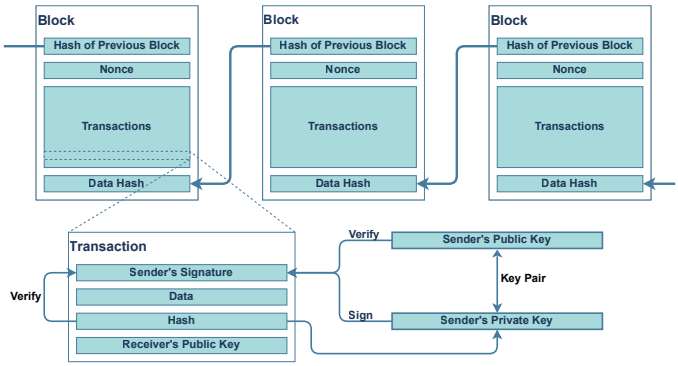


Fig. 1: Components of a block and transaction in a general Blockchain network

Figure 1 shows the components inside a block. One block contains: hash of its previous block, which points to the last block in the chain before it was pushed; nonce, which is a number to indicate the order of the block in the sequence; a set of valid transactions; and the hash of the current block, which will be referred to when there is a new block afterwards.

Ethereum is a Blockchain implementation. Similar to Bitcoin, the Ethereum Blockchain also has its cryptocurrency called Ether. The difference that makes Ethereum outstanding is that it allows a block to store executable programs. This kind of application is called Smart Contract. A Smart Contract allows the execution and the storage of the program stage to be performed in a distributed and decentralised way. The operation codes (opcode) in the Smart Contract were designed to be Turing complete, which means that it can execute any program algorithms that today's computers can perform (22). Similar to the other Blockchain implementations, all the nodes in the Ethereum network contain the same transaction data, including Smart Contracts and their contract states. This concept is like having a computer whose instances are distributed over the Blockchain network. When a contract method is called, the node will add the method calling transaction into the chain. The transaction can be interpreted and executed to update the contract state. When the transaction is propagated over the network, the other nodes will also update their contract state in the same way. This distributed machine is called Ethereum Virtual Machine (EVM).

Figure 2 shows the workflow of an Ethereum Smart Contract. The top level programming language to write a Smart Contract can be either Solidity or Vyper, which have similar

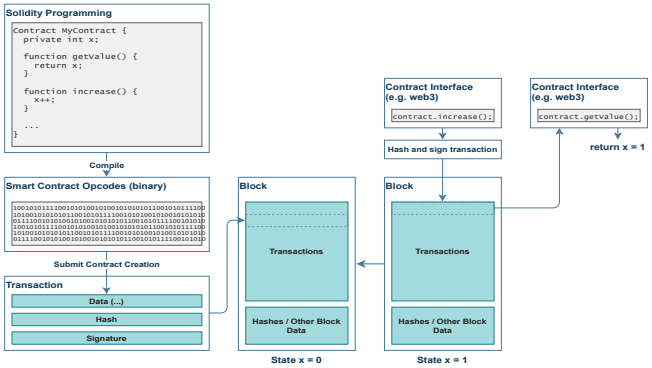


Fig. 2: An example flow of contract creation and function calling in Ethereum Smart Contract

syntaxes to JavaScript and Python, respectively. After that, the programming codes are compiled into Ethereum opcode binaries. An opcode indicates a computational operation of the EVM like an opcode in a computer program. The data of these binaries are then embedded into a transaction in a block and pushed into the chain. Figure 2 also shows that the calling method which changes the contract state (by altering the value of the variable in the contract) needs to be submitted as another transaction in another block. However, if the method is read-only (returns the contract state value without updating it), it can be called instantly without being submitted as another transaction. Calling a method in Smart Contracts needs to be paid for by its cryptocurrency. The price of calling a method is determined by the gas spent in the calculation. The concept of gas is similar to the electricity consumption in a physical computer. Each opcode in a Smart Contract spends a different amount of gas determined by Ethereum. The maximum amount of gas is limited by the Ethereum network. For this reason, a method that has too many operations, especially iterations, is likely to cause an out-of-gas error from the Ethereum network.

C. Spatial Indexing

A computer processes and handles data in binary. Therefore, the data that are not based on binary integers require more complicated data structures and different standards to work with, for instance, decimal (floating) point, number, text, picture, sound, as well as geospatial data. The performance and efficiency of querying and accessing these types of data can be improved by using indexing techniques. An indexing technique constructs the desired data in such a way as to make them a particular kind of searchable keywords. When a user wants to carry out a query for the data, a lookup table containing the sorted indices can be used to quickly look for the position where the desired data are located. However, indexing geographical data is more complicated as it is multidimensional and generally related to Euclidean space (23). There are different ways to index geospatial data, for example, R-Tree, which is based on a binary search tree for querying a spatial object (24). This thesis will focus on the

geocoding-based indexing techniques to store and query for the spatial data objects in the Blockchain.

Geocoding is the conversion of a geospatial object into another kind of interpretation. Some geocoding techniques aim to ease human readability, the case of a postal address, for example. On the other hand, some of them aim to increase machine-readability because the geocoded value results in a binary integer. Examples of this type of geocoding technique include Geohash and S2, which are going to be studied in this work. However, geocoding techniques are not two-way compatible. This means that the geocoded information cannot be reverted to the exact original geospatial object, but only to a similar one (25). Nevertheless, loss of accuracy is tolerable in this work, as its aim is not to store the exact geospatial data, but to use them as additional contextual information for reputation management in an IoT system.

III. Location-Based Reputation Management System

IV. A systematic architecture

The reputation management system of IoT devices is based on the cloud-fog-edge architecture. The cloud layer is generally in charge of storing and analysing data from edge sensors, as well as controlling, visualising data and interacting with users. Then, the fog layer contains a number of devices that can be geographically distributed. Each fog device will be a node in the Blockchain network. They will connect to each other and form an Blockchain network *to serve the Geospatial Smart Contracts* for managing location-based reputation indices of end devices. As a Blockchain node, a fog device will be associated with a number of geographical regions. For example from Figure 3, fog *W*, *X*, *Y*, *Z* are associated with regions *W*, *X*, *Y*, *Z*, respectively.

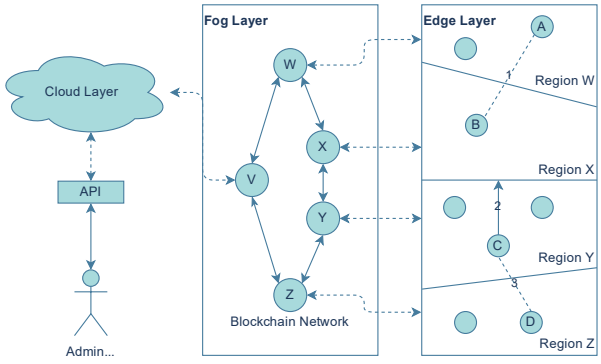


Fig. 3: Overall architecture of the proposed reputation management system

Then, in the Edge layer, an end device is equipped with sensors or actuators and communicates with other end devices in order to consume the services (dashed-line 1 from Figure 3). Service consumption between devices is based on their trust, which they can decide on by using the reputation index stored in the Smart Contracts from the fog layer. Therefore, an edge device needs to communicate with the fog layer to interact with the contracts in order to discover a service provider in the target area as well as to query its reputation value. Furthermore, devices in the proposed system are assumed to be movable and can be displaced across the different areas. Hence, when a service provider device enters a different area it should have a different reputation value because it is not recognised in the region. For example, from Figure 3, device *C* can move from region *Y* to *X* (arrow 2), but once it has moved, device *D*, which is its consumer, should no longer recognise its reputation. And to consume the service it would establish a new connection with another device in the area that it can trust.

The location-based reputation management system plays a role in the IoT system when an end device has moved. The related interaction flows are divided into three categories, which are 1) region management flow, which happens when there is a fog layer that wants to associate itself to a geographical region in the system, 2) edge device mobility flow, which happens when an edge device has moved within the same or between different regions, and 3) reputation management flow, which happens when an edge device wants to consume a service from another device.

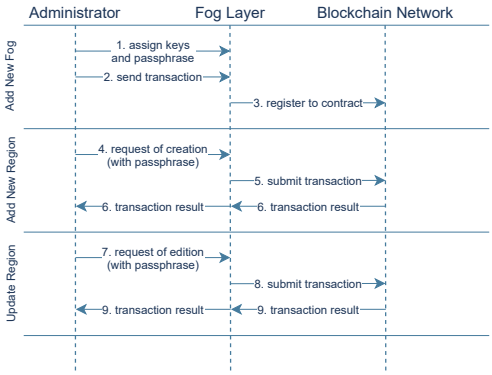


Fig. 4: The workflow of region management in the system

The first interaction flow, which is region management flow, happens in the initial stage when setting up the fog devices and their associated geographical regions, as well as when there are changes in those regions. Figure 4 shows this flow in diagram form. To register a new fog device in the network, the administrator needs to assign a public address for the fog device (arrow 1) and use the private key to sign the transaction for calling the registration method of the Smart Contract (arrow 2, 3). Once the fog device has been registered, it can assign itself to an encoded geographical region, so that those edge devices in the region will have reputation values handled by this fog device. To either add a new region or update an existing region of a fog device, the administrator will interact with the device (arrow 4, 7) to authenticate it with the private key, and the device will then submit the signed transaction to the Blockchain network (arrow 5, 8). Finally, the Blockchain network returns the result of the transaction back to user (arrow 9).

The second workflow is on the edge layer side, which is

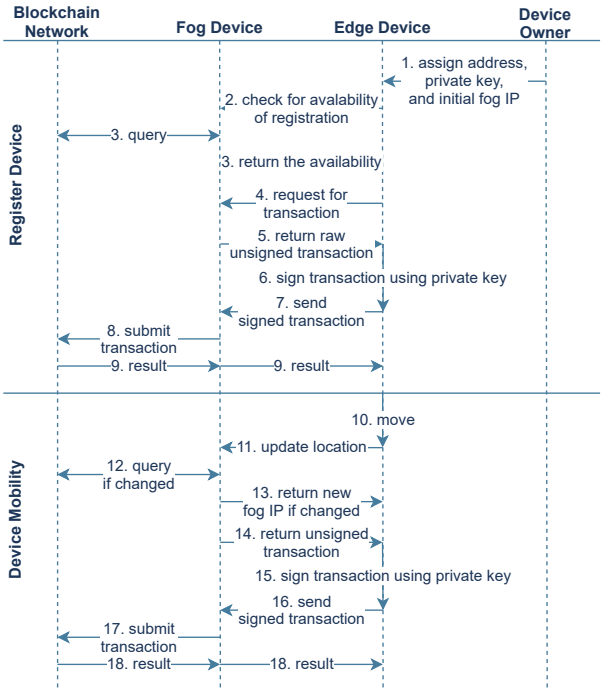


Fig. 5: The workflow of edge device mobility

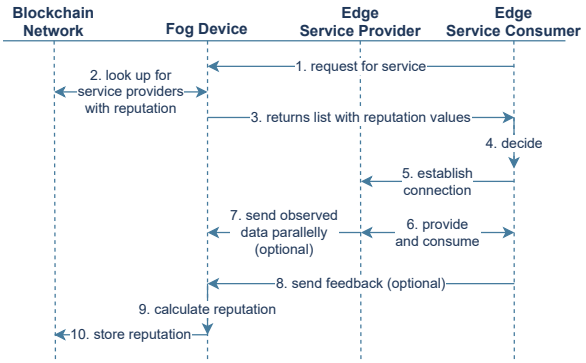


Fig. 6: The workflow of reputation query and generation

represented in Figure 5. It involves the registration of an edge device in the network and the consequence when the device moves. To register a new device, the owner must assign a key-pair to the device and give the address of the fog layer so that the device can communicate with the network (arrow 1). The edge device will then check with the fog layer whether it can be registered (arrow 2, 3). If it is able to do so, the edge device will request registration from the fog layer (arrow 4). To authorise the action, the fog layer will give an unsigned transaction back to the edge device (arrow 5). The edge device will then use the private key that has been assigned to sign the transaction and submit it to the network (arrow 6, 7, 8). Then, the network returns the result of transaction submission back to the edge device (arrow 9).

When a registered device moves (arrow 10), it will check with the fog layer whether its new location falls within a different region (arrow 11, 12). If so, it must update both the address of the fog device that it is communicating with (arrow 13) and its data in the Blockchain network by using its private key to sign the given transaction (arrow 14, 15, 16). Then the fog device passes that signed transaction to Blockchain network for submission, and the network returns back the result as the final step (arrow 17, 18). By this way, the fog layer cannot update the edge device information in the Blockchain network because the private key of the edge device is not shared.

The last event is when a device wants to consume a service. Figure 6 shows the flow of this interaction. The consumer requests a service from the fog layer. The fog node handles it by looking up the results in Smart Contracts (arrows 1, 2). It then returns a list of candidate service providers and their reputation values in the area back to the consumer (arrow 3). The consumer considers this list and establishes a new connection with the provider it trusts the most (arrows 4, 5). While the connection is ongoing (arrow 6), in parallel the service provider can send the data to the fog device (arrow 7), which can later use these data to calculate the service quality as a factor for generating a new reputation index. After the connection has finished, the consumer can send its feedback regarding the service to the fog node (arrow 8). The fog layer finally has both sensory data from the provider and feedback from the consumer and thus it can use this information to calculate a new reputation index for the service provider (arrow 9) and update the value in Smart Contracts (arrow 10).

V. Geospatial Smart Contract

Figure 9 shows the class diagram of the proposed Geospatial Smart Contracts. There are three main contracts in the proposed architecture. The first one is the *Regions* contract. The contract manages the regions and their associated geospatial areas. It has methods for adding a geometry from a list of geocoded cells (*addCells* function *cellIDs* parameter) or from tree-structured data (*addTree* function *data* parameter). It also has the *query* function for finding the geographically associated region by giving a geocoded cell ID. The second contract is the *Devices* contract. This contract manages edge devices registered in the system. It has the *registerDevice* function for a new edge device in the system. The edge devices can also call the *updateDeviceIPs*, *updateDeviceLocation* and *updateDeviceServices* functions to update the device information. These functions can only be called by the devices themselves, as they validate callers by their public address, and the valid transaction to call these functions can only be signed with the private key. The contract also has additional methods for obtaining and searching for devices in the system. It can look up a device by giving a public address through the *getDeviceFromAddress* function. It can also look for a list of devices that are located in a given region by the *getDevicesInRegion* and *getDevicesInRegionWithService* functions, or it can be given a location and then look for the devices that are located in the same region with the location through the *getDevicesInSameRegionWithService* function. Finally, the last contract is the *ReputationManagement* contract, which joins the *Regions* contract and *Devices* contract and handles

the storage and querying of reputation values of the devices registered in each region. The contract allows the corresponding fog device to update the reputation value of a service provided by an edge device in a specific region through the *updateReputation* function. The function can only be called by the fog device that is associated with the target region, and it is validated by the public address and the signature of the function caller. The contract also has the *getReputationValue* and *getReputationData* functions for querying the reputation value and reputation history records, respectively, of a service provided by an edge device that is also in the given region.

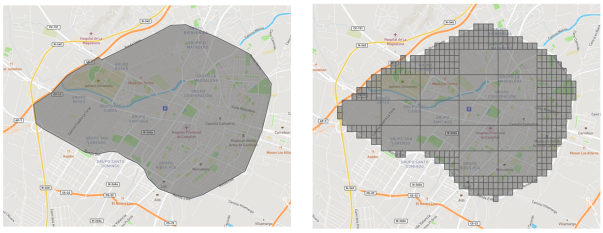


Fig. 7: Example of a polygon covered by geocoded cells

As described in the previous sections, this research work decided to use geocoding techniques to store the geographical areas (polygons) of the associated regions in the Smart Contracts to manage the reputation and perform a spatial query. Despite the different encoding algorithms implemented in the two techniques selected (*Geohash* and *S2*), both of them result in a binary integer whose length indicates the hierarchical level of the cell. Therefore, they can share most Smart Contract methods. So they are designed to inherit from the same abstract contract and they override only the methods that work differently depending on the technique selected.

Inside the regions contract, a geographical area will be geocoded into a combination of cells previously stored in the contract as shown in Figure 7. The left image shows the original polygon of the region and the image on the right displays the group of geocoded cells that are combined to fit the original polygon. Because of this, the binary representation of both geocoding techniques is hierarchical, that is, it can merge the group of cells that fulfil the lower level into one cell in the upper level. This behaviour can be observed from the image on the right in Figure 7.

Figure 8 shows another example of a geocoded region. Part *(A)* of the figure shows the geocoded cells and their textual identification in Geohash. There are 11 cells from 3 different levels in this example. Part *(B)* of the figure lists these cell IDs. When a letter consumes 1 byte in the memory, this list of Geohash cells will consume at least 64 bytes excluding cell separation such as a new-line character. From this list it can be observed that there is redundancy of the data, especially the prefix of those cells. Therefore, this work proposes another data structure for storing this group of geocoded cells by using the tree approach. Part *(C)* of Figure 8 shows the tree-based structure of this geocoded region. The nodes in blue represent the cells that have children, while the nodes in purple represent the final level that will be included in the region data. This

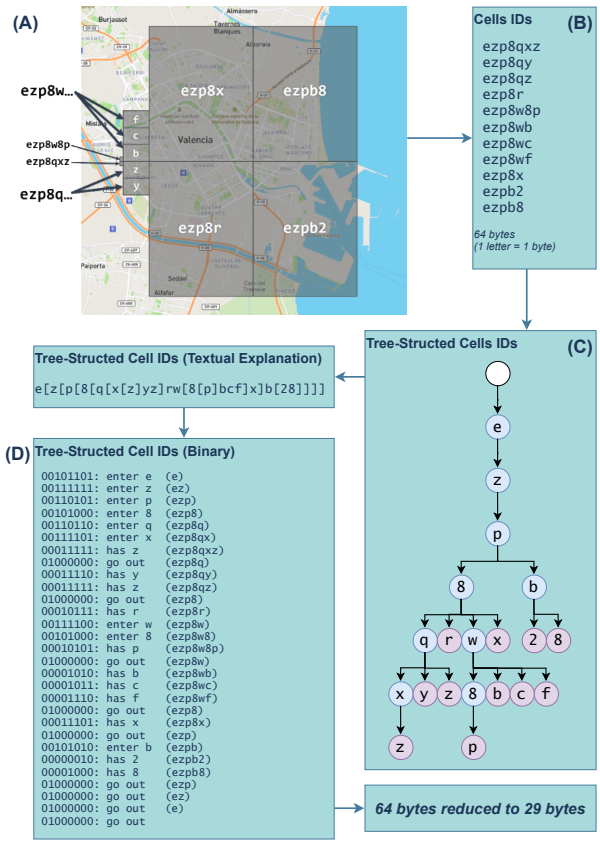


Fig. 8: An example of the proposed geocoded cells compression technique

tree structure can be encoded into the binary representation shown in part *(D)* of the figure. Geohash cells use base32 representation, which requires only 5 bits to store one level and, hence, storing a level in one byte of data will result in 3 bits remaining unused. These 3 bits can therefore be used to store additional information to indicate whether the node cell has children or not. The result from binary encoding the tree in part *(C)* requires only 29 bytes to store the same data, which is less than half of the original data, which requires 64 bytes. In the same way, an S2 cell list can also be compressed by the same algorithm. However, an S2 cell uses 2 bits for one level, and storing one 2-bit level in one byte is not sufficient as there will be 6 bits left over for each node. Therefore, the S2 cell will be grouped with a length of 6 bits before being transformed into a tree. The spare 2 bits are used for marking whether the level is open or closed in a similar way to Geohash.

VI. Implementation Details

This work focuses on the fog and edge layers. Although the cloud layer has been considered, no details are given of its implementation. To be specific about the components inside each layer that makes the architecture functional, firstly, in the fog layer, there are two components. On the one hand, there is the Blockchain network using Ethereum, which stores smart contracts of information on the managing device and the reputation data. The second component is an API, which is a

communication interface for interaction between the administrators or edge devices and the smart contracts. The Ethereum network will be deployed using Go Ethereum (Geth), which is an open-source implementation of the Ethereum Blockchain network.

Fog API is another component inside the fog layer that connects users to Smart Contracts. A user of this API can be either an administrator who manages the system, or an edge device that requests the discovery of a service provider or queries the reputation value of a device. The API uses an HTTP protocol to accept the requests and returns responses in a JSON format, while at the same time it also communicates with Ethereum to serve the requests. It also performs a preliminary condition check before calling Smart Contract in order to avoid an unexpected failure transaction. The interaction interface of the Fog API is divided into a number of different controllers, which are listed in Figure 10. Each controller serves a different purpose and interacts with different users in the system. Most of the controllers require a connection to the Blockchain in order to query or submit a transaction to the smart contracts.

The API is developed in JavaScript language run in the NodeJS engine using the HTTP protocol[2]. The reason for using JavaScript as the programming language is that it can communicate with the Geth client using the web3.js library. The library also allows interaction with a Smart Contract to be performed in a simpler way through Application Binary Interface (ABI) of the contract. Additionally, while developing the Smart Contracts, it uses Truffle, which is a programming suite designed for Smart Contract development. Truffle contains multiple tools that allow the developed contracts written in Solidity to be compiled, tested and deployed in the Ethereum network. Each fog node serves an API using the HTTP protocol, while it also communicates with the Geth instance through internal channel at a different port. Therefore, the API does not have to store the private key of the current node, but it can access the Blockchain through Web3 API using a passphrase given by the key owner. Through this, when a verified user wants to send a valid transaction through the fog node, the user must attach the correct passphrase in its HTTP request header, the API uses this passphrase to decrypt the private key and unlock the account.

When a user uses the API to call a Smart Contract method, the API needs to submit the contract call transaction to the Blockchain network, which is done by *web3.js* library. However, in some cases, it might take a longer time for the Blockchain node to mine a block and propagate the transaction. In consequence, waiting too long for the transaction result might be interrupted by a request timeout response from the HTTP connection. The user then receives an error even the transaction might be correctly pushed to the Blockchain in next few minutes. To tackle this problem, in the developed API, when a user calls a controller that submits a new transaction to the Blockchain, after *web3.js* processes the request and submitted the transaction for mining, it returns the transaction hash of the contract method call as a response instead of waiting for the transaction to be mined. A user can then use the hash with another endpoint in the API to check the transaction status.

The last layer is the Edge layer. An edge device is implemented using Particle Development Boards[3]. The development board has a similar specification to Arduino, but the Particle board allows for the possibility of compiling and flashing the programme to the board using its cloud service. The programming part of the board is written in C++ language. As previously described, the communication between edge devices will be performed by the API on HTTP protocol. Hence, the library that serves this propose is ParticleRdWebServer[4] by *robdobsn*. Lastly, the board needs to be able to sign the transactions using the ECDSA algorithm so it will use the *micro-ecc*[5] library provided by *kmackay* to do so. The service provider and consumer code in the edge device were then developed[6]. The *ParticleRdWebServer* library allows a service provider to serve simple requests from its client.

VII. Experimentation and results

This section describes the experiment designs and procedures, as well as their evaluation. It is divided into three experiments. The first defines the procedures and criteria for comparing the two geocoding techniques and the proposed tree-based data structure. The second experiment describes the simulation of the proposed architecture to test the different aspects and the last experiment describes the implementation methodologies and the test case scenario.

A. Experiment 1: Comparison of Geocoding Techniques

This experiment is designed to compare the *Geohash* and *S2*[7] geocoding techniques. The comparison will be performed by: a) covering or fitting a GeoJSON polygon into a list of geocoded cells (using data on the administrative regions in Spain), and b) compressing geocoded cells using the proposed algorithm. In each step, it compares the results in terms of their output size and calculation time. There are two issues that need to be addressed in this comparison experiment. Firstly, the popular Geohash is based on base32 representation, despite the fact that its binary notation allows the level to increase by 2 bits, as S2 does, but the common Geohash libraries support only base32 manipulation. Hence, increasing one level in a Geohash cell adds 5 more bits, but it adds 2 bits in the case of S2. The consequence of this is that there are no such levels where these two techniques provide similar accuracy, and hence they cannot be compared fairly. Table I shows the comparison of cell area sizes between Geohash and S2 at different levels.

[2]GitHub repository: https://github.com/ponlawat-w/uji_mt-fog_api

[3]https://www.particle.io/

[4]https://github.com/robdobsn/ParticleRdWebServer/

[5]https://github.com/kmackay/micro-ecc

[6]GitHub repository: https://github.com/ponlawat-w/uji_mt-edge_devices

[7]The GitHub repository of the code in this experiment is available at *Geohash:* https://github.com/ponlawat-w/uji_mt-geohash_evaluation_test *S2:*https://github.com/ponlawat-w/uji_mt-s2encoding

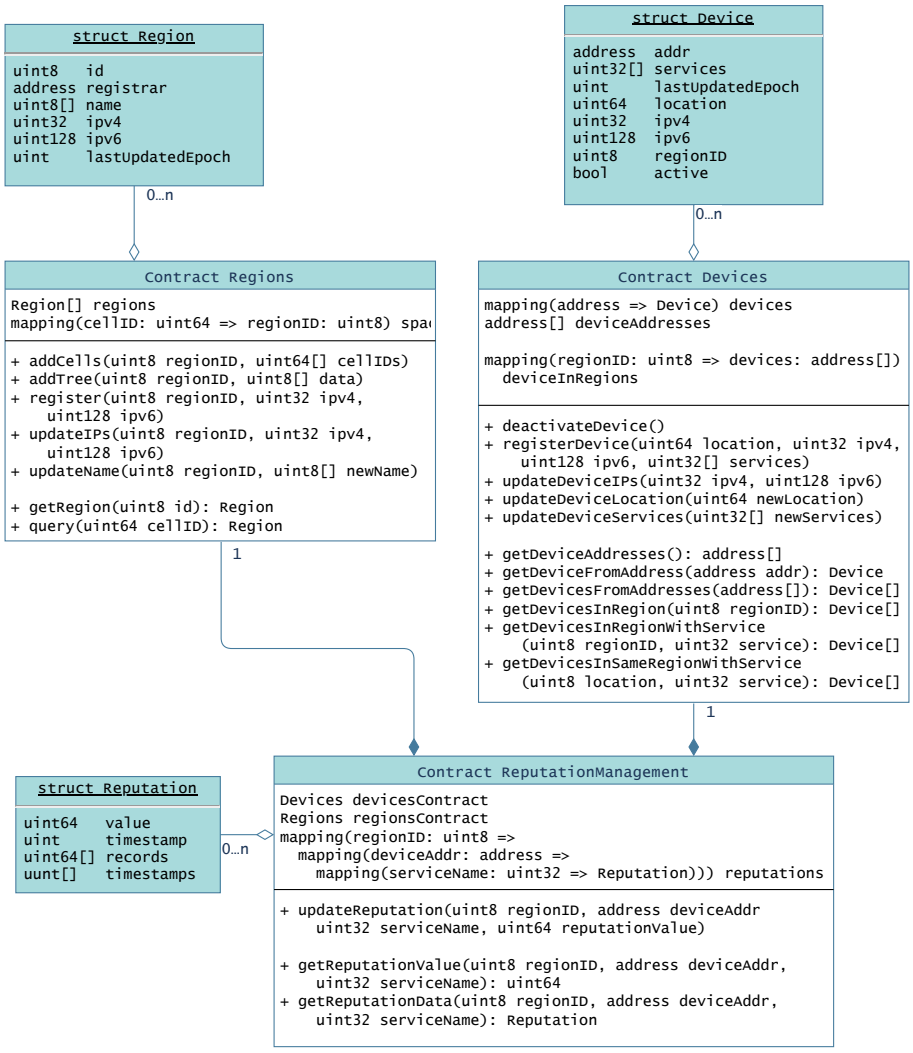


Fig. 9: The extended class diagram of Smart Contracts to define the Geospatial Smart Contract

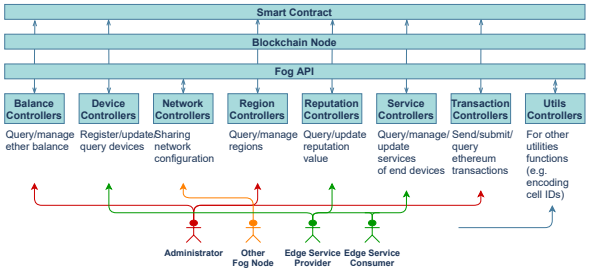


Fig. 10: Controllers and interaction diagram of Fog API

From the table (Table I) it can be observed that at the same bit length, Geohash has a smaller cell size, which means it provides greater precision than S2 at the same length. We can note that the number of S2 bits shown in the table includes the end bit of 1, which is insignificant for interpreting the coordinates. For this reason, before comparing these two techniques it is necessary to decide on the factor for pairing the levels, for example, whether they are to be based on data length or cell size. Since the smart contracts in this work store geocoded cells in a fixed-length integer of 64 bits, the number of bits stored is not significant regardless of its level. It therefore compared the levels whose area sizes are most similar, which are: Geohash level 4 and S2 level 9, Geohash level 5 and S2 level 12, Geohash level 6 and S2 level 14, and Geohash level 7 and S2 level 16.

Figure 11 shows the size in kilobytes of the resulting geocoded cells, using the administrative region polygons in Spain as the input data. The horizontal lines, both black solid and dashed, are sizes of the original polygons in Well-Known Binary (WKB) and GeoJSON, respectively. The row CompressedGeohash and S2 shows the file sizes of the cell ID lists in each technique. The rows GeohashTree and s2base64tree show the file sizes of the compressed cell ID lists using the proposed algorithm. The Graph is shown in the logarithm scale. The result shows a very similar file size between Geohash and S2 for preserving a similar precision. It also shows that the proposed tree data structure can greatly reduce the size from the geocoded cells list. It also demonstrates that using compressed Geohash level 6 or S2 level 14 can save more disk space than the original polygonal data, but for the next level

Geohash	Bit	Cell Size	S2	Bit	Cell Size
1	5	12,588,175.24 km^2	0	4	85,011,012.19 km^2
			1	6	21,252,753.05 km^2
			2	8	6,026,521.16 km^2
2	10	786,760.95 km^2	3	10	1,646,455.50 km^2
			4	12	413,918.15 km^2
3	15	12,293.14 km^2	5	14	104,297.91 km^2
			6	16	26,113.30 km^2
			7	18	6,529.09 km^2
4	20	768.32 km^2	8	20	1,632.45 km^2
			9	22	408.12 km^2
5	25	12.01 km^2	10	24	102.03 km^2
			11	26	25.51 km^2
			12	28	6.38 km^2
6	30	0.75 km^2	13	30	1.59 km^2
			14	32	0.40 km^2
7	35	11,723.65 m^2	15	34	99,638.93 m^2
			16	36	24,909.73 m^2
			17	38	6,227.43 m^2
8	40	732.73 m^2	18	40	1,556.86 m^2
			19	42	389.21 m^2
			20	44	97.30 m^2
9	45	11.45 m^2	21	46	24.33 m^2
			22	48	6.08 m^2
10	50	0.72 m^2	23	50	1.52 m^2
			24	52	0.38 m^2
11	55	111.81 cm^2	25	54	950.23 cm^2
			26	56	237.56 cm^2
			27	58	59.39 cm^2
12	60	6.99 cm^2	28	60	14.85 cm^2
			29	62	3.71 cm^2
13	65	0.11 cm^2	30	64	0.93 cm^2

TABLE I: Maximum error (cell size) of different levels in the Geohash and S2 Geocoding Techniques

both techniques consume a larger amount of space than the WKB data.

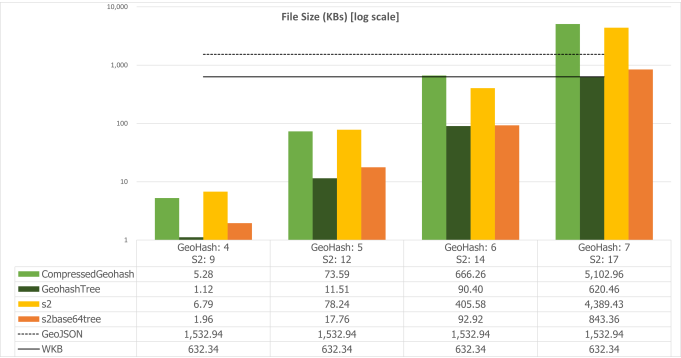


Fig. 11: Graph showing a comparison of the file size (in kilobytes) in the Geohash and S2 techniques

Figure 12 shows the number of cells outputted from the Geohash and S2 geocoding techniques. The graph is displayed using a logarithmic scale. From the graph, it can be observed that Geohash and S2 resulted in a similar number of cells, but it is not clear which one is better, as S2 contains a greater number of cells on the lower level, while at the higher level Geohash requires more. The reason could be the characteristics of the input data, such as the alignment of the input polygons, as some polygonal shapes might fit in one technique better than another.

Figure 13 shows the calculation time spent on geocoding the polygons using the Geohash and S2 techniques. The results

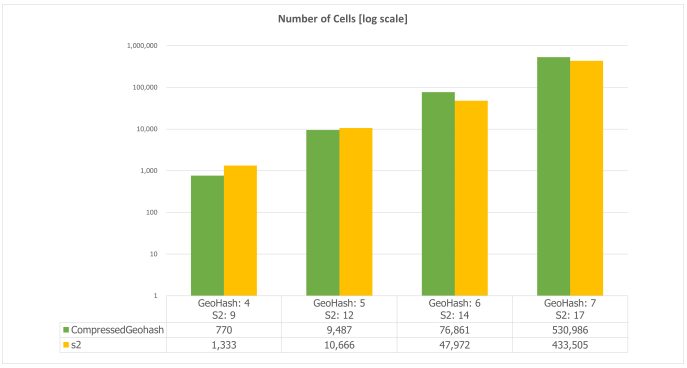


Fig. 12: Graph showing the number of cells resulting from fitting areas to geocoded cells using the Geohash and S2 techniques

from the two techniques are very different. The y-axis has to be split, where the left (blue) axis indicates the Geohash calculation time in seconds and the right (orange) indicates the S2 calculation time in the same unit. From the result, it can be observed that S2 is much faster than Geohash, as converting the whole areas took less than 10 seconds even at the highest precision, while Geohash took more than ten minutes to accomplish all the tasks. However, the reason for this difference could be that they used a different programming language. The Geohash experiment uses JavaScript and the S2 experiment uses GoLang due to limitations of the supported library. Furthermore, the library used to fit the polygons into the S2 cells is the official library developed by the S2 developer team, but the Geohash library is developed by a third-party developer.

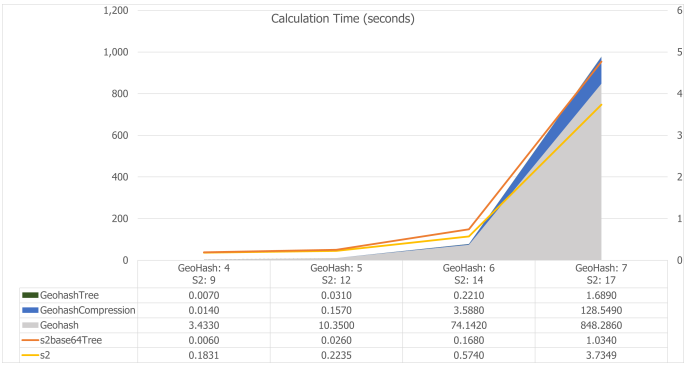


Fig. 13: Graph showing the calculation time for geocoding and compressing the data using the Geohash and S2 techniques

B. Experiment 2: Contract simulation

The second experiment was to develop Smart Contracts based on the geocoding techniques of the proposed architecture and test their performance. The procedure in this experiment was to measure the following: a) input size and time spent on adding regions using Geohash and S2, based on either a list-structure or a tree-structure; b) time spent on querying for a cell in the Smart Contract based on Geohash and S2, and c) time spent on mining a block in an Ethereum Blockchain using a personal computer and a fog device (Raspberry Pi).

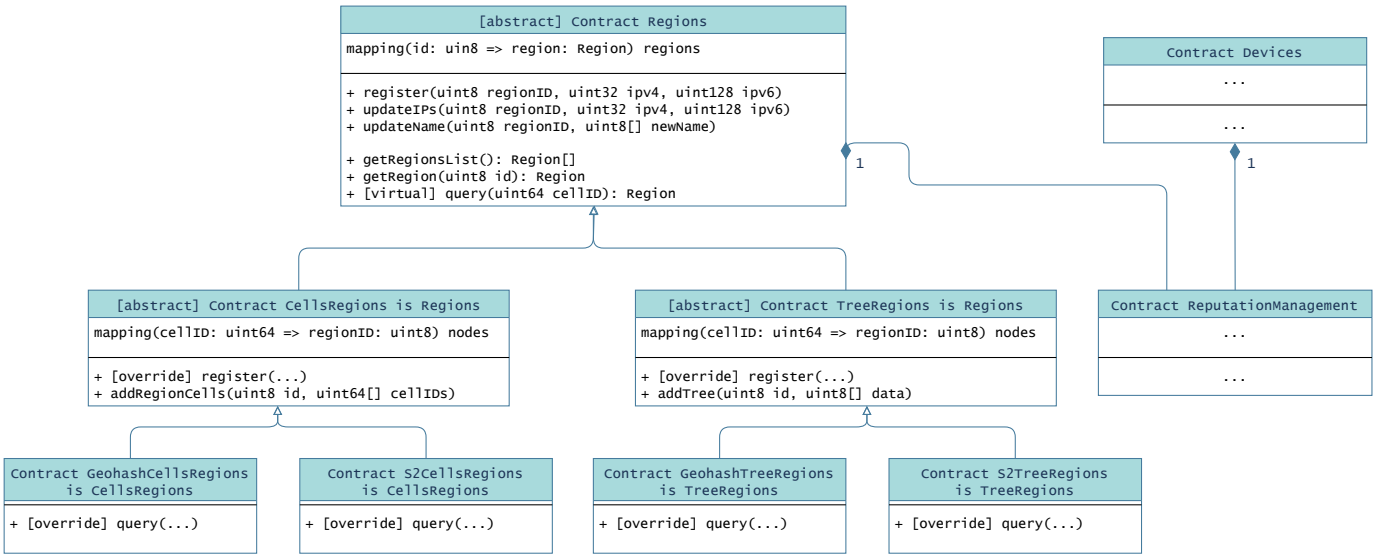


Fig. 14: Updated UML class diagram of Smart Contracts for supporting both geocoding techniques

For this experiment, the previously designed Smart Contracts were developed. Figure 14 shows the diagram developed for this experiment. The regions contract became an abstract contract because its structure and geocoding technique needed to be defined. So the contract was inherited by a CellsRegions contract for the list-based structure and by a TreeRegions contract for the tree-based structure. It was then inherited again following either the Geohash or the S2 geocoding techniques.

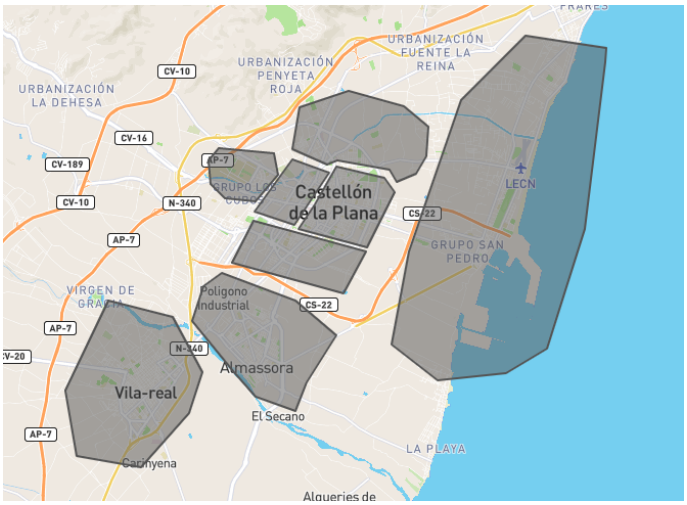


Fig. 15: The regional divisions of data used in the simulation experiment

Figure 15 shows the regional divisions of the data used in this experiment. The locations used in querying and device mobility assessment in this experiment are randomly generated within these regions.

The design of this experiment divided data inputs into three different levels. In the case of Geohash, the maximum cell lengths are 6, 7 and 8. And in the case of S2, they are 14, 17 and 19. Figure 16 shows the input data lengths of the different techniques and data structures. The result is in line with the first experiment, where the sizes of the tree-structured input data are smaller than the list-structured data. In addition, regarding the geocoding technique, we cannot tell exactly which technique is better because the results vary depending on the level of precision.

Next, Figure 17 shows the gas consumption when adding the regions in Smart Contracts. From the results, it can be noticed that, despite having smaller input data, tree-structured contracts consume more gas than list-structured ones. A possible reason could be that the tree-structured ones require more computation to expand the tree before assigning cells to the region, unlike the list structure, where the contract can iterate over the members and directly assign them to the region. However, regarding the geocoding technique, the result is similar to the input size, where it varies in different levels of precision. In the same way, the result of time spent has a similar trend with the gas consumption, as shown in Figure 18.

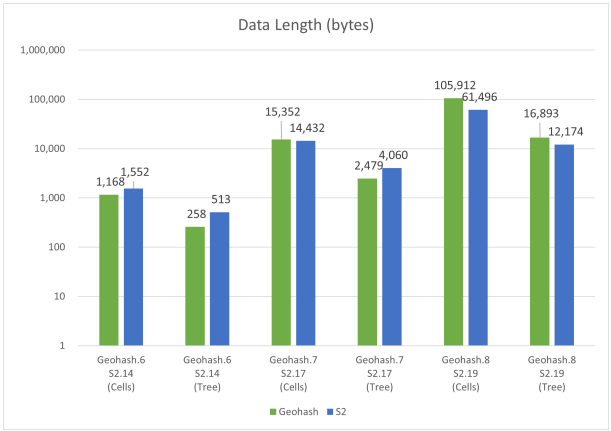


Fig. 16: Graph showing the sizes of the input data used for adding new regions to the contracts

The last point to be tested is the query behaviour. 6,000 points

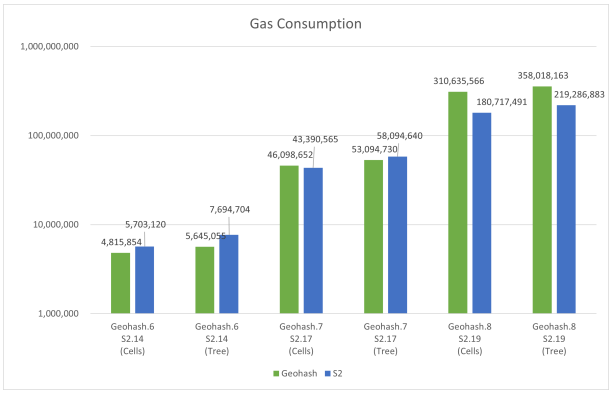


Fig. 17: Graph showing gas consumption when adding new regions to the contracts

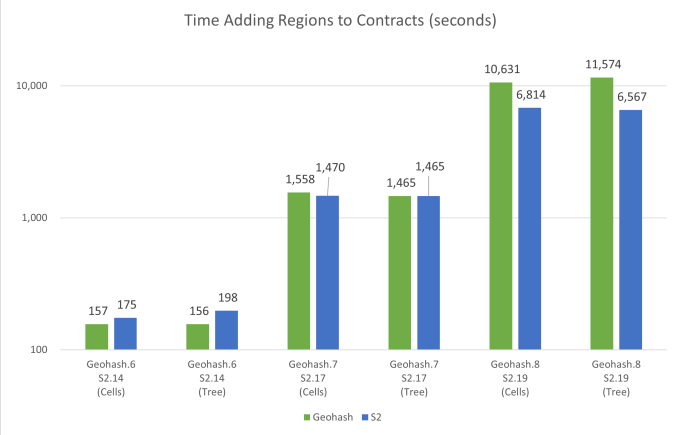


Fig. 18: Graph showing time spent when adding new regions to the contracts

were randomly generated in space and the same data set was used to test the query time in the Smart Contracts based on Geohash and S2. Of these 6,000 points, 5,000 belonged to an existing region in Smart Contracts, while the other 1,000 were the points that did not fall into any region. Figure 19 shows a box plot of the query time. From the results, it can be observed that there is no difference between querying data in Smart Contracts in terms of geocoding technique or level of precision.

Lastly, this experiment was designed to compare the possibilities and performance of the Ethereum nodes using IoT devices when mining a new block to the chain. It compares a personal computer with a 4-core CPU and a Raspberry Pi, which is used as a fog device. However, due to hardware limitations, the Raspberry Pi did not manage to mine any blocks. Instead, it returned an out-of-memory error and halted the Ethereum client. For this reason, despite the fact that Raspberry Pi can be a node in the Ethereum network and can submit the transactions, it cannot be a miner to publish the chain. A possible solution to this issue is to modify the implementation of the Ethereum network so that, instead of the default consensus algorithm, Proof-of-Work, it uses the Proof-of-Authority algorithm. However, this experiment continued to

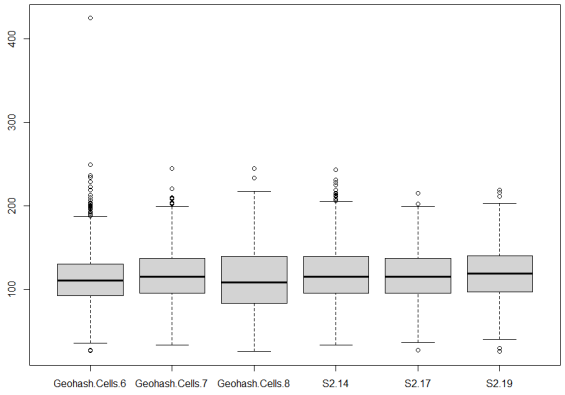


Fig. 19: Graph showing time spent when querying for a cell in the contracts (in milliseconds)

measure the mining time using only the personal computer running on the different thread numbers and keeping the default consensus algorithm, that is, Proof-of-Work. Figure 20 shows the results. As expected, using more threads tends to take less time to mine a block into the chain. The mining time indicates the time needed to publish a transaction in the network. Hence, from the results it can take up to minutes for a change in Smart Contract to be propagated over the network.

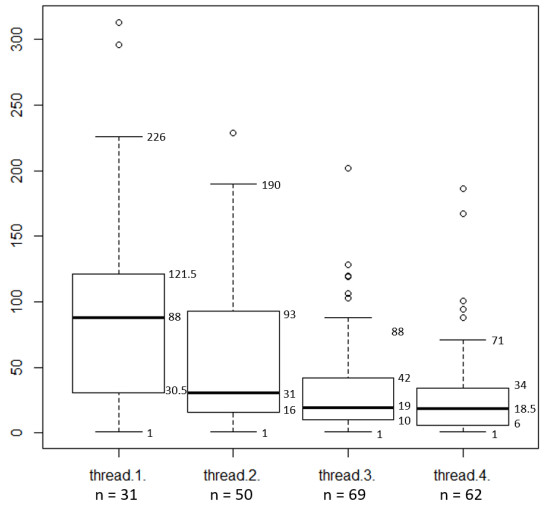


Fig. 20: Graph showing time spent when mining a new block in the chain (in seconds)

C. Experiment 3: Deployment and Scenario

The last experiment is designed to answer the last objective by proving that the proposed architecture works in IoT devices by deploying the program in both fog and edge devices. The fog devices used in this experiment are a *Raspberry Pi 4 Model B*, while the edge devices are *Particle Development Boards*.

The experiment set the program developed up in the devices and used the described workflow to test the communication between them. This experiment was expected to confirm that: a) A fog node is able to serve as an Ethereum node; b) A service consumer is able to discover the providers available and their reputation data, as well as provide feedback after the consumption; and c) A service provider is able to provide the service and to sign the transaction.

The APIs and a Geth client were installed in a Raspberry Pi. After using the API to interact with Smart Contracts, it showed that the Raspberry Pi had enough potential to serve as an API and run an Ethereum node at the same time. The Raspberry Pi, whose operating system is based on Linux, is also able to run another service, including SSH, which allows remote access to the device. Hence, it is not necessary to be with the node physically in order to maintain, update or configure it.

Smart Contracts allowed the devices to register themselves with their IP address information. Once it is added into the Blockchain, another node can also see the list of the registered fog nodes and their IP addresses, so that they can use the IP address as a URL base for the API. However, in this test scenario, the registered IP addresses are in the same LAN network. In practice, the fog node can be installed in a different area that uses a different internet network. The IP address or using a VPN enables the device to be accessible from outside. Figure 21 shows the setup of this test experiment. The Raspberry Pi that acts as a fog node is located as a black box on the right-hand side of the image. In the test scenario, the edge service provider was attached to a temperature and humidity sensor, and exposed its device registration data via services. The device's location had to be simulated and set manually by the user, which is not supposed to be the case in the practical implementation.

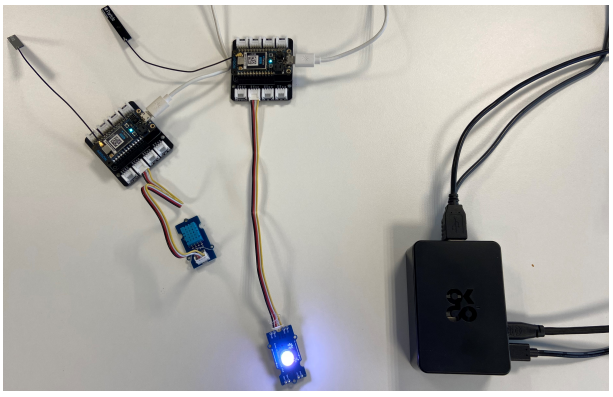


Fig. 21: The setup of the test scenario: edge service provider, edge service consumer, fog Raspberry Pi

After the device had been switched on, it was able to receive an Ethereum address and private key from the user, and register itself to Smart Contracts through the fog API. To send the transaction, the device managed to sign a transaction and submit it to the fog API. By using an HTTP caller application in the personal computer (in this case *Postman*), it was able to communicate with the edge device through its API served on the exposed IP address. The IP address of the device can be discovered by querying the contract via the fog API. It was also able to query for the reputation value using the fog API and to establish a new connection to consume the service. The service provider returned correct values of the service and, in parallel, sent the observed data to the fog node as defined in the architecture. Finally, after finishing the service consumption, the user sent feedback to the fog node, which was later calculated (or random in this work) to be a new updated reputation value of the service, and updated in Smart Contracts. When a user queried the reputation value again, the smart contract returned the updated value as expected.

The service consumer device is equipped with an LED that can emit light in red-green-blue shades of colour. The device was programmed to display its communication status via LED, which is red when it cannot connect to the fog node, orange when there is no satisfying reputed service provider in the area, blue when it is establishing a new connection with a service provider, and green when it is consuming the service correctly. The result shows that the service consumer worked correctly. It emitted an orange signal when the queried area had no service providers whose reputation value satisfied the threshold, and it emitted a green one when there was a service provider that was trusted and selected to establish a new connection, as shown in Figure 22.

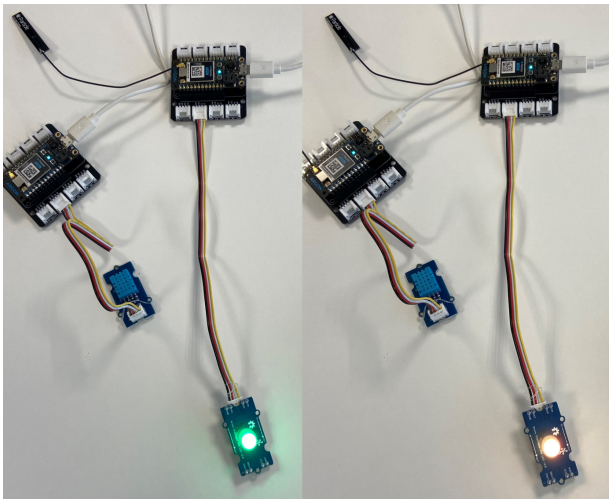


Fig. 22: The service consumer status LED shown on green (trust) and orange (no trust)

VIII. Conclusion and Future Work

This research work has proposed an architecture for a reputation and device management system in IoT, based on the cloud-fog-edge architecture. The architecture uses device location to determine its associated area, which is a factor in managing the reputation data. The management system is implemented in the fog layer in a decentralised way by using Blockchain technologies. The work adopted Ethereum as a Blockchain network because it provides a Smart Contract that allows an executable program to be stored in the Blockchain network. The Geospatial Smart Contracts store the spatial data of associated areas for discovering and querying

registered services and their reputation values. For complexity and performance reasons, a polygon object that represents an area is encoded using hierarchical geocoding techniques: either Geohash or S2. This work also proposed a compression algorithm aimed at reducing the geocoded data.

The first experiment in this work compared the Geohash and S2 geocoding techniques. Despite a few concerns regarding the comparison, the experiment results showed that S2 took less time to encode a polygon into a list of cells. In terms of data size, there was not much difference between the two techniques. The proposed tree structure showed a satisfactory outcome, because it can reduce the data size considerably. The second experiment implemented the proposed architecture and tested it with a large number of random input data. The results showed that the Geospatial Smart Contract were able to function as expected to manage the reputation values and device service discovery in a decentralised way. The experiment also showed that, even though the tree-structure has a smaller data size than the list-structured data, it consumes more Ethereum gas when added to the contracts because it needs to make more effort to expand the tree. However, once the data have been added to the contracts, there were no big differences regarding query time. The experiment also revealed that Raspberry Pi was unable to mine a block in the Ethereum network using the default Proof-of-Work as a consensus algorithm due to its hardware limitations. This means that, without changing the Blockchain consensus algorithm, a bigger computer is needed as another node in the fog layer in order to keep the Blockchain network functional. The last experiment deployed the architecture in IoT devices by using Particle boards as the edge devices and Raspberry Pi as the fog device. The results demonstrated that, as a fog node, the Raspberry Pi was able to serve an API service and interact with Smart Contracts. In the edge layer, the service provider was able to serve its service and sign the transaction to interact with the Smart Contract through the fog layer. At the same time, the service consumer was also able to communicate with the fog layer in order to discover a provider with its reputation data, and was able to make a decision to consume the service based on the reputation data.

During the development of this work, several problems and limitations were detected. An unexpected problem that emerged during the implementation regarding the limitation of JavaScript was that this language does not support bit-wise operation for an integer with a length of more than 32 bits (26). The reason is that although the language stores numbers in 64 bits, the binaries are in the IEEE 754 floating-point standard, not in a general integer, although the number does not have a decimal point. And the normal integer, which does support bit-wise operations, is in only 32 bits. This issue leads to problems when working with the geocoded information whose data are stored in a 64-bit integer and need bit-wise operations to manipulate the data. This issue can be tackled by using an external library or splitting every data into two parts - each 32-bits. Fortunately, there were no intensive bit-wise operations in the proposed algorithms; hence, for the sake of simplicity, the data were converted into hexadecimal numbers as a textual string, and handled by JavaScript built-in string functions.

Another problem encountered during the experiment is that Ethereum limits the maximum amount of gas allowed to be spent in a transaction. Therefore, sending a large amount of input data to Smart Contract is likely to be rejected because it exceeds the gas limit. To solve this issue, those input data need to be split into a number of smaller sized chunks, and send multiple transactions instead.

For the last problem, the architecture designed in this work uses the spatial context to query and manage devices and their reputation data. Therefore, an edge device is expected to be movable and, consequently, equipped with a positioning module, for example, a GPS module. However, due to the environment limitation of the university laboratory, the time constraint of the thesis, and the steps taken to avoid working physically due to the ongoing pandemic at the time of this work, it was not possible to obtain a positioning module for the edge devices to test the scenario of the architecture. Thus, the service consumption API in the provider device exposed another endpoint to let the user set its location manually. In a practical implementation, this API endpoint is expected to be replaced by a position reading function from a GPS module. Then, the location is expected to be geocoded into a Geohash or S2 cell, depending on the technique used in Smart Contracts.

The research results showed that the proposed architecture was functional, either in a simulation mode within a single machine, or in an IoT system using distributed devices. A fog device can be an Ethereum node to handle the running of the Smart Contract. There was some delay in the mining of a transaction as a new block over the network, which is an expected disadvantage of Blockchain. Hence, using the Blockchain network might not be suitable for a real-time system or a system that updates data frequently. In addition, not only a fog device, but an edge also showed that it can be a part of the Blockchain network, despite its limited computational ability, because it was able to perform the ECDSA calculation to sign a transaction using an assigned private key. This ability in the edge device allows the proposed architecture to be more secure, because even the fog node that submits the transaction for edge devices cannot have its data tampered with as long as the private key is not known.

Furthermore, the results also showed that a spatial context can be added to the smart contract. The contract in the system was not designed for storing an exact original polygon so, instead, the data were geocoded before being added to the Smart Contract. A trade-off of using geocoding techniques is the loss of precision, because the geocoded data are not able to represent the original area exactly. This drawback affects the accuracy when querying a point near the border of an area. However, the main objective of this work is not to store the same geographical object in Smart Contracts, but to use it as a spatial context for managing and querying data. Hence, this error is acceptable. Furthermore, the geocoded data can also be sorted and indexed in a data lookup table, so it can easily and quickly find out whether a cell is inside a

particular region or not, without this involving any geometric calculation. However, due to the fact that this work adopted and compared two different geocoding techniques (*Geohash* and *S2*) to represent the geographical data in the system, and the results do not show a big difference between them, either of them can be used in the proposed architecture. For the current scale, Geohash might be more suitable as it is relatively more legacy, and it might be easier to find a supporting library and communities for development. On the other hand, S2, which is somewhat newer, is also an interesting option for further development and for taking more advantage of its potential as elaborated in the introduction section.

As future work, there are several possibilities for further development. Firstly, this work proposed only a management architecture, but not the calculation of the value. The information collected from the devices in this work can be used in future work to generate a real reputation value of the service providers, such as by using a machine-learning technique to detect abnormalities in the sensory data, or to use the feedback from the consumers as a factor to calculate the value. There is vulnerable information transmitted between devices in the proposed architecture, which is the Ethereum private key when assigning a new device, or the passphrase when communicating with the fog API. To prevent the data from being eavesdropped by a third person, in practice, it is suggested to use a secure HTTP channel (HTTPS) to encrypt the messages in the communication.

Additionally, this work proposed an abstract architecture for managing the reputation values that can be applied in other IoT applications. In practical implementations, communication between devices in a system can follow a common IoT standard, such as *Mozilla WebThings*[8] or *OGC SensorThings*[9]. The integration between the architecture and these standards needs to be studied in the future to make the proposed concept more practical.

Regarding the geocoding techniques, the experiments showed that both geocoding techniques are suitable for the current work. However, related work shows that S2 performs better in different aspects. Therefore, the usage of S2 and its further spatial computation can be studied in greater depth for future work. In addition, even the tree-structured input data consume more gas and take longer to process in Smart Contracts, but future work could take advantage of its reduction in data size.

Finally, this research work demonstrated the possibility of manipulating spatial data in a decentralised application. Despite the limitations of Smart Contract compared to a traditional application, it is worth studying further more advanced spatial data manipulation in a decentralised context. To be more precise, this work only used the geocoding technique to query the spatial data. Future development could extend this part and perform a different spatial calculation, for example a range query, region adjacency, intersection, or handling other types of spatial data, such as points or lines.

[8]https://iot.mozilla.org/

[9]https://www.ogc.org/standards/sensorthings

ACKNOWLEDGEMENTS

This study was supported by the TRUST4IoE project of the Programa Estatal de Proyectos de I+D de Generación de Conocimiento of the Spanish government (grant number PID2019-104065GA-I00). Sergio Trilles has been funded by the postdoctoral Juan de la Cierva fellowship programme of the Spanish Ministry for Science and Innovation (IJC2018-035017-I).

REFERENCES

[1] C. Granell, A. Kamilaris, A. Kotsev, F. O. Ostermann, and S. Trilles, "Internet of things," in *Manual of digital earth*. Springer, Singapore, 2020, pp. 387–423.

[2] S. Trilles, A. Luján, O. Belmonte, R. Montoliu, J. Torres-Sospedra, and J. Huerta, "Senviro: A sensorized platform proposal using open hardware and open standards," *Sensors*, vol. 15, no. 3, pp. 5555–5582, 2015. [Online]. Available: https://www.mdpi.com/1424-8220/15/3/5555

[3] G. M. Køien, "Reflections on trust in devices: an informal survey of human trust in an internet-of-things context," *Wireless personal communications*, vol. 61, no. 3, pp. 495–510, 2011.

[4] L. Mui, M. Mohtashemi, and A. Halberstadt, "A computational model of trust and reputation," in *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, 2002, pp. 2431–2439.

[5] G. Lenzini, M. S. Bargh, and B. Hulsebosch, "Trust-enhanced security in location-based adaptive authentication," *Electronic Notes in Theoretical Computer Science*, vol. 197, no. 2, pp. 105–119, 2008, proceedings of the 3rd International Workshop on Security and Trust Management (STM 2007). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1571066108000595

[6] S. Trilles, J. Torres-Sospedra, Ó. Belmonte, F. J. Zarazaga-Soria, A. González-Pérez, and J. Huerta, "Development of an open sensorized platform in a smart agriculture context: A vineyard support system for monitoring mildew disease," *Sustainable Computing: Informatics and Systems*, vol. 28, p. 100309, 2020.

[7] Cisco, "Unleash the power of the internet of things," *Cisco Systems Inc*, 2015.

[8] J. Golosova and A. Romanovs, "The advantages and disadvantages of the blockchain technology," in *2018 IEEE 6th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*, 2018, pp. 1–6.

[9] R. Deiotte and R. L. Valley, "Comparison of spatiotemporal mapping techniques for enormous etl and exploitation patterns." *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, vol. 4, 2017.

[10] Z. Balkić, D. Šoštarić, and G. Horvat, "Geohash and uuid identifier for multi-agent systems," in *Agent and Multi-Agent Systems. Technologies and Applications*, G. Jezic, M. Kusek, N.-T. Nguyen, R. J. Howlett, and L. C. Jain, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 290–298.

[11] J. Chen, Z. Tian, X. Cui, L. Yin, and X. Wang, "Trust architecture and reputation evaluation for internet of things," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 8, pp. 3099–3107, Aug 2019. [Online]. Available: https://doi.org/10.1007/s12652-018-0887-z

[12] J. Guo, I.-R. Chen, D.-C. Wang, J. J. P. Tsai, and H. Al-Hamadi, "Trust-based iot cloud participatory sensing of air quality," *Wireless Personal Communications*, vol. 105, no. 4, pp. 1461–1474, Apr 2019. [Online]. Available: https://doi.org/10.1007/s11277-019-06154-y

[13] M. Debe, K. Salah, M. H. U. Rehman, and D. Svetinovic, "Iot public fog nodes reputation system: A decentralized solution using ethereum blockchain," *IEEE Access*, vol. 7, pp. 178 082–178 093, 2019.

[14] D. E. Kouicem, A. Bouabdallah, and H. Lakhlef, "An efficient architecture for trust management in ioe based systems of systems," in *2018 13th Annual Conference on System of Systems Engineering (SoSE)*, 2018, pp. 138–143.

[15] F. Victor and S. Zickau, "Geofences on the blockchain: Enabling decentralized location-based services," in *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, Nov 2018, pp. 97–104.

[16] K. Ashton, "That 'Internet of Things' Thing," *RFID Journal*, 2009.

[17] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645 – 1660, 2013, including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13000241

[18] L. Atzori, A. Iera, G. Morabito, and M. Nitti, "The social internet of things (siot) – when social networks meet the internet of things: Concept, architecture and network characterization," *Computer Networks*, vol. 56, no. 16, pp. 3594 – 3608, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128612002654

[19] D. Artz and Y. Gil, "A survey of trust in computer science and the semantic web," *Journal of Web Semantics*, vol. 5, no. 2, pp. 58 – 71, 2007, software Engineering and the Semantic Web. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570826807000133

[20] M. Swan, *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015.

[21] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Satoshi Nakamoto Institute*, 2008.

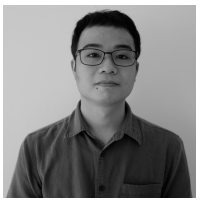
[22] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, 2014.

[23] H. Lu and B. C. Ooi, "Spatial indexing: Past and future," *IEEE Data Eng. Bull.*, vol. 16, no. 3, pp. 16–21, 1993.

[24] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '84. New York, NY, USA: Association for Computing Machinery, 1984, p. 47–57. [Online]. Available: https://doi.org/10.1145/602259.602266

[25] R. Moussalli, M. Srivatsa, and S. Asaad, "Fast and flexible conversion of geohash codes to and from latitude/longitude coordinates," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, 2015, pp. 179–186.

[26] Mozilla Developer Network, "Expressions and operators - JavaScript — MDN," https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators, 2020, [Online. Accessed on 30 January 2021].



Ponlawat Weerapanpisit graduated with a Bachelor's Degree in Computer Engineering from Chiang Mai University, Thailand, and an Erasmus Mundus Joint Master's Degree in Geospatial Technologies (2019-2021) from the Universitat Jaume I, Spain, Universidade NOVA de Lisboa, Portugal, and the University of Münster, Germany. His master's thesis dealt with decentralising location-based reputation management systems in IoT using Smart Contracts.

Sergio Trilles received his PhD in the Integration of Geospatial Information from the Universitat Jaume I in 2015. He is the author of more than fifty peer-reviewed journal and conference publications. He has had the opportunity to work as a researcher for four months in the Digital Earth and Reference Data Unit of the European Commission's Joint Research Centre (JRC). He is currently a postdoc researcher in the GEOTEC group.

Joaquín Huerta is a full professor in the Department of Information Systems at the Universitat Jaume I in Spain, where he teaches several courses related to GIS and Internet Technologies. His current research interests are indoor positioning, smart cities, mobile and web GIS applications and augmented reality. He is the head of the GEOTEC Research Group, Director of the Erasmus Mundus Master of Science in Geospatial Technologies degree programme, which is run jointly with the Universities of Münster and Nova de Lisboa. He is and has been the principal investigator of several research projects, including EU projects such as A-WEAR, GEO-C, EUROGEOSS. In addition to academic activities, he is also a founding member of UBIK Geospatial Solutions http://ubikgs.com.

Marco Painho is a Professor of Geographic Information Systems and Science at the Nova School of Information Management (NOVA IMS) of the Universidade Nova de Lisboa, Portugal. He holds an undergraduate degree in Environmental Engineering from the Universidade Nova de Lisboa, a master's degree in Regional Planning from the University of Massachusetts, Amherst, and a PhD in Geography from the University of California, Santa Barbara. He is the coordinator of the Master's Degree in Geographic Information Systems and Science and the Master's Degree in Science in Geospatial Technologies. He has over 30 years of experience in the GIS domain and has coordinated over 100 projects in the application areas of the environment, natural resources management transportation and teaching, among others. He is the author and editor of over 200 academic and professional publications.