SenseChain: A Blockchain-based Crowdsensing Framework For Multiple Requesters and Multiple Workers

Maha Kadadha[a,*], Hadi Otrok[a], Rabeb Mizouni[a], Shakti Singh[a], Anis Ouali[b]

[a] *Electrical and Computer Engineering Department, Khalifa University of Science and Technology, Abu Dhabi, UAE*
[b] *Emirates ICT Innovation Center (EBTIC), Abu Dhabi, UAE*

Abstract

In this paper, we propose a decentralized crowdsensing framework for multiple requesters with multiple workers built on Ethereum blockchain- *SenseChain*. Crowdsensing frameworks utilize workers' sensing capabilities to fulfill requesters' sensing tasks. While crowdsensing is typically managed by a centralized platform, the centralized management entails various challenges such as reliability in workers' selection, fair evaluation for payment distribution, potential breach of users' privacy, and high deployment cost. The proposed solution, *SenseChain*, is a decentralized crowdsensing framework developed to run on Ethereum blockchain to mitigate said challenges while increasing users' engagement, with reasonable cost. *SenseChain* is developed around three smart contracts: 1) *User Manager Contract* (UMC), 2) *Task Manager Contract* (TMC), and 3) *Task Detailed Contract* (TDC). These contracts implement the platform features such as maintaining users' information, publishing tasks from multiple requesters, accepting reservations and solutions from multiple workers, and evaluating solutions to calculate proportional payments. The framework is implemented using Solidity and Web3.js, where a real publicly available dataset is used. The framework performance is compared to a centralized greedy selection framework to demonstrate its comparability while mitigating tackled challenges. The results in terms of solutions quality, time cost, and workers traveled distance confirm its viability as a solution for crowdsensing.

Keywords: Crowdsensing, Blockchain, Smart Contracts, Reputation, Decentralized

1. Introduction

Background and Motivation The mass ownership of devices with integrated sensors led to a shift from traditional data collection to crowdsensing/ sourcing [1]. Crowdsensing/ sourcing frameworks such as Amazon Mechanical Turk (MTurk) [2] and Uber [3] utilize

*I am corresponding author

Email addresses: maha.kadadha@ku.ac.ae (Maha Kadadha), hadi.otrok@ku.ac.ae (Hadi Otrok), rabeb.mizouni@ku.ac.ae (Rabeb Mizouni), shakti.singh@ku.ac.ae (Shakti Singh), anis.ouali@ku.ac.ae (Anis Ouali)

workers' devices to collect required data and fulfill tasks. Their objective is to facilitate completing tasks with the best possible quality while motivating users to frequently engage with the platform [4]. In MTurk, requesters are individuals and companies such as Raytheon, Apple, etc [5], who request tasks for a fee. Workers are registered users, approximately 100K by the time of writing this paper [4], interested in fulfilling these tasks. Typically, a trusted centralized platform governs the crowdsensing activity as it eases the management of multiple requesters with multiple workers. It holds users' information (reputation), collects tasks, selects workers, evaluates their submissions, and shares their payments.

Centralized crowdsensing frameworks have been proposed [6, 7, 8, 9, 10, 11, 12, 13] with different objectives. However, two main limitations exist due to their centrality: 1) lack of execution visibility, which affects the trust in the framework's decisions; i.e. worker selection, data evaluation, payment calculation, or information update (reputation), 2) high deployment and maintenance cost, where service fees are enforced on users. These limitations lower users' utilities, which in turn diminishes their interest to engage with the framework [1].

Distributed crowdsensing frameworks such as [14, 15, 16, 17] are proposed to overcome the problems of centralized frameworks while focusing on different objectives such as data analysis, task selection, or sensing group formation. These efforts provide adequate performance with their distributed execution. Nevertheless, the trust issue remains unsolved as members of the frameworks are assumed to be trusted, which is not always the case.

Ethereum blockchain [18] emerged as a trusted decentralized infrastructure used in different areas such as Vehicular Ad hoc NETworks (VANET) [19], and Cloud [20]. Blockchain [21] is a distributed immutable ledger of transactions, organized as blocks, confirmed by independent miners in the network for a fee. The Ethereum blockchain extends on blockchain by permitting the autonomous execution of smart contracts [22]; programs deployed with built-in functions. The autonomous execution of smart contracts enables trusted, transparent, and immutable execution for transactions.

Blockchain-based crowdsensing/sourcing frameworks [23, 24, 25, 26, 27, 28, 29] have been proposed to eliminate the need for a trusted centralized platform by migrating its functions to Ethereum smart contracts. However, the main limitations in these blockchain-based existing frameworks are:

- Dependency on evaluation functions from requesters, which may be biased to reduce the quality of workers' submissions, their payments, or reputations.
- Lack of penalization for misbehaving requesters to identify them in future tasks.
- Focus on one requester scenario [27] and studying the feasibility of frameworks without comparing their performance to centralized models [24, 25, 26, 27].

Consequently, the main issue this work proposes to tackle is: *How to design a decentralized blockchain-based crowdsensing framework for multiple requesters with multiple workers? The framework should be trusted, ensures fair evaluation, provide trusted feedback about users, while resulting in comparable performance to centralized models with reasonable cost.*

Proposed Model In this paper, a blockchain-based decentralized crowdsensing framework is proposed- *SenseChain*. It overcomes the need for a trusted centralized platform in

crowdsensing frameworks by utilizing the decentralized Ethereum blockchain as its underlying infrastructure. It brings together requesters and workers in a collaborative sensing platform that does not require pre-established trust between parties. *SenseChain* implements the functions of the centralized framework in smart contracts such as user registration, task collection from requester and allocation to users, solution submission and evaluation for workers' payments. The smart contracts guarantee distributed-like characteristics such as unbiased transparent evaluation of submissions, lack of single point of failure, and trusted immutable information.

In *SenseChain*, users create their own accounts and Ethereum addresses without relying on a trusted third party to maintain them. Then, their Ethereum addresses are used to interact with other members in the framework. In the proposed framework, users' addresses, types, and reputations are maintained and updated by the *User Manager Smart Contract* (UMC) deployed in the blockchain for reliability. Requesters create tasks by adding them to the *Task Manager Smart Contract* (TMC) which consequently creates the corresponding *Task Detailed Smart Contract* (TDC). TDC is responsible for collecting reservations from workers, determining their *Quality of Information* (QoI) to approve them, and evaluating solutions to calculate workers' payments. The evaluation of task solutions in TDC is executed by miners making it immune to manipulation from both the requesters and the platform. The consensus of multiple miners ensures the trust in the outcome of the evaluation.

The smart contracts are implemented in Solidity [30] and the framework is deployed on a local private blockchain and tested using a real dataset. The performance is compared to a centralized greedy worker selection model [6]. Both frameworks show comparable performance in terms of quality of selected workers, time cost, workers' traveled distance, and submitted solution quality, while requiring reasonable cost.

Therefore, the contribution of this work is a decentralized blockchain-based crowdsensing framework for multiple requesters with multiple workers which:

- Utilizes the proposed smart contracts running on the Ethereum blockchain to manage interactions between requesters and workers.
- Calculates trusted reputation values for requesters and workers with fair feedback.
- Evaluates submitted solutions in an autonomous and transparent manner to incentivize or penalize requesters and workers accordingly.
- Overcomes limitations of centralized platforms with comparable performance and reasonable cost.

Organization The remainder of this paper is organized as follows. Section 2 presents the problem statement. Section 3 presents the background and related work. Section 4 presents the proposed *SenseChain* framework. Section 6 presents the evaluation of the proposed framework. Section 7 presents *SenseChain*'s cost analysis. Section 8 concludes the paper.

2. Problem Statement

A crowdsensing model entails three main entities: requesters, workers, and a platform, typically centralized. Requesters are users who publish sensing tasks to the platform while specifying a termination criteria such as deadline or number of responses required. In addition, they specify a defined budget they are willing to pay as an incentive for workers [31] to motivate them into fulfilling the task. Workers find pending tasks and submit to the ones they are interested in doing for a payment in return. A platform acts as an intermediary between requesters and workers as it collects published tasks and selects the set of workers to perform these tasks. Once a task is completed, the platform evaluates its submissions, shares proportional payments, and updates the reputation of users accordingly. A platform would log users' interactions for reputation calculation to represent their reliability.

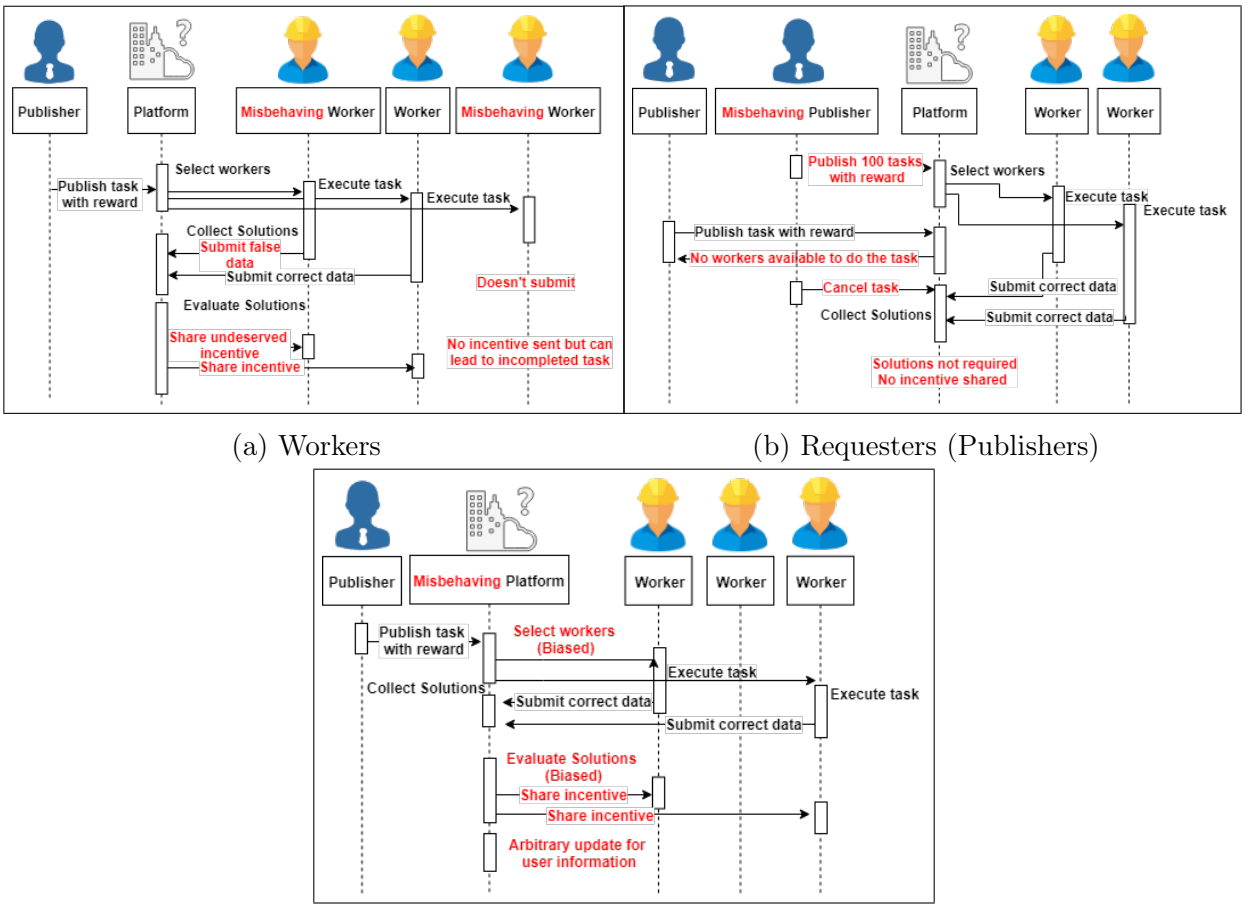
However, these entities can compromise the security of the framework by misbehaving in an active or passive manner, which threatens the viability of the crowdsensing activity. Passive misbehavior is when an entity attempts to collect unauthorized information without affecting the execution of the platform such as the Man-in-the-Middle attack. Active misbehavior is when an entity intentionally performs acts to harm other members in the framework. In this work, we tackle cases of active misbehavior by the different crowdsensing members: workers, requesters, and the platform as illustrated in Fig. 1.

Misbehaving Workers mainly compromise the quality of completed tasks by their acts. As shown in Fig. 1a, they can intentionally submit incorrect data biasing the evaluation, hence receive undeserved payments. Furthermore, these workers can submit multiple solutions to the same task using different identities to bias the evaluation and favor their submitted solution. Hence, increase their acquired payments. The impact is even more evident when a few participants are involved and the majority of them misbehave. In our case, the payments of honest workers are drastically reduced. Additionally, misbehaving workers may commit to a sensing task but never submit a solution, which leads to having incomplete tasks due to lack of submissions, thus harming requesters.

Misbehaving Requesters are of greater impact as they are assumed to be honest with no penalization enforced by the platform. As shown in Fig. 1b, these requesters can compromise the availability of the framework by issuing redundant tasks, that make workers unavailable for legitimate tasks from other requesters. In addition, they can cancel their published tasks after workers commit to them to not pay deserving workers who have already completed the tasks.

Misbehaving Platforms, shown in Fig. 1c, are f great concern, specially with the increased awareness among users about their consequences. As centralized platforms' execution process is hidden from users, they can alter the selection of workers to favor some workers over others. In addition, misbehaving platforms can alter users' information to reduce their reputations, which affects workers' future selection to perform tasks. Furthermore, misbehaving platforms can alter the evaluation of submitted solutions to reduce payments.

Misbehavior of platforms can be seen in commercial ones such as Amazon Mechanical Turk (MTurk) [2]. In fact, MTurk is facing fraud accusations recorded by workers who claim that requesters reject their submissions without paying them [32], where MTurk does not



(a) Workers (b) Requesters (Publishers)

(c) Platform

Figure 1: Crowdsensing Misbehavior Models

validate requesters' decisions for fairness. Furthermore, MTurk does not provide any feedback mechanism to identify these misbehaving requesters and record workers' objections [33]. Hence, there is a need to mitigate these issues and overcome the limitations of centralized management to provide a more trusted framework. Unfortunately, none of the existing works handle the existence of untrusted platforms despite their impacts.

Therefore, the problem this work tries to handle is: *How to design a decentralized crowdsensing framework which handles the three misbehaviour models when multiple requesters and multiple workers interact with the platform? The framework should provide fair evaluation and trusted feedback about users while resulting in comparable performance to centralized models with reasonable cost.*

3. Background and Related Work

In this section, the background on blockchain and smart contracts is summarized. Then, existing distributed crowdsensing/ sourcing frameworks are presented. In addition, blockchain-based crowdsensing/ sourcing framework are also described.

3.1. Blockchain and Smart Contracts

Blockchain [21] is a distributed immutable ledger of transactions organized as blocks and maintained by distributed peers [34]. The transactions are executed and confirmed by independent users called *miners* in the network for a reward. Miners verify transactions according to the selected mining mechanism (i.e. proof-of-work, proof-of-stake) before digitally signing and appending them to the chain. Blockchain utilizes two cryptography techniques: 1) Elliptic Curve Cryptography (ECC) and 2) Secure Hash Algorithm (SHA-2) to verify the authenticity and integrity of data. Blockchain's design allows for decentralization, traceability, execution and data transparency while preventing data and execution tampering. Transparency implies that transactions and data are propagated to all members in plain text, which means that privacy is not maintained. On the other hand, compared to centralized platforms, blockchain maintains a degree of identity privacy as users use their own generated public and private keys, which their Ethereum addresses are derived from. Nevertheless, recent works have shown that deanonymization of users' identities is possible as Ethereum addresses can be linked to the public IP addresses used by the users [35]. That opens the door for identity exposure with additional techniques being deployed making the exposure independent from the use of blockchain.

Bitcoin [34] is the first digital cryptocurrency used in an application on top of the blockchain infrastructure. Bitcoin is used for currency exchange and requires miners to determine the valid block with its set of transactions using the Proof-of-Work (PoW) consensus mechanism. In PoW, a miner needs to solve a computationally expensive cryptography puzzle by finding a hash of multiple input metrics including the hash of the previous block. Considering the previous hash links transactions in a stronger manner, thus improving the immutability and trust of the executed transactions.

Ethereum blockchain [18] emerged as an extension of the Bitcoin blockchain by allowing execution of built-in programs known as smart contracts [22]. This blockchain is a decentralized platform of thousands of Ethereum Virtual Machines (EVM), which autonomously, transparently and irreversibly execute the smart contract functions without the need of a trusted third party. Different from blockchain, Ethereum deploys the keccak256 hashing algorithm to verify transactions. A smart contract presents the digital agreement between parties predefined and implemented as rules and functions executable on the EVM. Functions are triggered by users' interactions or scheduled events, where each computational step endures a cost paid by the triggering party. The computational cost is measured as *gas consumption* while being charged using the Ethereum blockchain cryptocurrency -*Ether*. Ethereum blockchain with its special smart contracts expands the application domain to Vehicular Ad hoc NETworks (VANET) [19], Cloud [20] as well as crowdsensing and -sourcing.

It is worth noting that blockchain comes with its own attacks which can be caused by misbehaving miners. Individually, miners are of negligible impact, however, a major attack is the 51% attack [36]. In this attack, 51% of the miners, their majority, consents on an illegitimate transaction (i.e. double spending in Bitcoin). Executing it successfully, while possible, requires an attacker to own huge computational resources, which is realistically hard to achieve. In this work, we do not mitigate misbehaving miners and these types of attacks are out of the scope of the paper.

3.2. Distributed Crowd Frameworks

As centralized crowdsensing frameworks can suffer from multiple limitations, distribution of these frameworks was introduced to overcome some of these limitations. Distributed crowdsensing frameworks [14, 15, 16, 17] are being applied for different objectives such as data analysis, task selection, or sensing group formation.

In [14], a distributed framework is proposed for users to select time-sensitive and location-dependent tasks. Each user determines the set of tasks it wants to complete and its mobility plan, through a task selection game. Despite that the task selection game runs in a distributed manner, the information required to run the game are acquired from the trusted service provider. Additionally, the service provider determines the quality of a user's submission and pays the user accordingly. However, the work assumes service providers and users share trusted information which is not always the case in a realistic environment.

In [15], a distributed data collection crowdsensing framework is proposed, which optimizes the cost of sensing in terms of energy. The framework allows data collection decision to be independently taken by each user. The cloud servers share the tasks for users to determine the choices that maximize their utilities. In addition, collected data are sent to the cloud for processing and analysis. While this work claims to be distributed, it still depends on centralized cloud servers to share, collect and analyze submitted data, which cannot always be trusted. Furthermore, the work focuses on minimizing the cost endured by users without considering the quality of their collected information.

In [16], a distributed data analytic model was proposed. The model reduces the load on back-end cloud servers by providing distributed analysis for submitted data at the fog level.

In [17], a distributed framework architecture- A3Droid is proposed for crowdsensing. This architecture allows distributed formation of sensing groups that can collect and manipulate data before sharing them with the cloud servers. The groups include a supervisor and followers where each have its own role.

While the above frameworks distribute aspects of crowdsensing, they assume members, users, fog nodes, or cloud servers, trust each other and do not misbehave. For example, users are allowed to select tasks they would like in a distributed manner and their decisions are assumed to be truthful even when done by independent members. Moreover, multiple limitations arise in these distributed frameworks such as users' privacy, consensus among the different members, and the trust between the members.

3.3. Blockchain-Based Crowd Frameworks

Multiple efforts have been proposed to integrate the blockchain technology into crowdsensing frameworks [23, 24, 25, 26, 27, 28, 29]. Blockchain enables running the crowdsensing framework in a decentralized manner while providing reliability; availability, and immutability. In this section, we present and discuss existing relevant models which deploy the blockchain technology to crowdsensing to understand their models and remaining challenges in the area.

The idea of incorporating blockchain to crowdsensing frameworks was first introduced in [23] where the of the Bitcoin network was proposed to autonomously pay incentives to workers' wallets as it preserves their identities. The proposed framework is said to be immune to attacks such as crowd sensor identity disclosure, certificate tampering, data

sensing tampering, and Sybil attack [37]. However, the proposed model still relies on a centralized address certification authority which defeats the purpose of utilizing Bitcoin network. As certificates are temporary, they require renewal once expired and migration of the user's reputation information between the certificates, which increases the overhead of the framework. This model was generally presented without evaluations for its performance.

On the other hand, CrowdBC [24] is a decentralized crowdsensing platform built on top of the Ethereum blockchain which guarantees the privacy of users while requiring low service fees and reliability in execution of the framework's functions through the deployed smart contracts. The framework requires deposits from both requesters and workers to prevent free-riding requesters and false reporting workers. However, while the model is decentralized and solutions' evaluation is done in the smart contracts, the evaluation function is still acquired from requesters. This makes the evaluation function dependent on a requester's reliability; a requester can submit a biased function that reduces the payments of legitimate workers. Requesters' penalization for cancelling tasks is distributing their deposits, which is not sufficient to distinguish them in future requests. Lastly, workers are required to query all users' contracts to find pending tasks which affects the scalability of the framework.

In [25], *ZebraLancer* was proposed as a blockchain-based crowdsensing framework which maintains privacy and allows interactions between requesters and workers. The main objective is to have the smart contract (hosted by the blockchain network) enforce the fair exchange between the submitted answers and their corresponding rewards without revealing any data or identity. The evaluation of solutions and calculation of incentives in this framework are conducted at requesters who are only required to post the proof for their calculations.

In [26], a privacy preserving incentive mechanism was proposed for crowdsensing where miners are utilized to verify the quality of sensed data and distribute reward among worker. Quality of solutions is calculated through the expectation maximization algorithm while privacy is implemented through the k-anonymity [38] to guarantee secure reward distribution. However, this model relies on the existence of a single server which could be considered untrusted with no punishment for misbehaving servers.

In [27], a blockchain-based crowdsensing framework was proposed where interactions between crowdsensing providers and users is orchestrated. The framework utilizes smart contracts to run an auction for user assignment to tasks where the selection of users is done according to their location while optimizing the payments of providers. The privacy of users is maintained in this framework by temporary accounts created by the Internet Service Provider (ISP) for users to interact with the tasks. However, this model assumes 1 requester only with multiple tasks where workers have limited preference. Moreover, the dependency on the ISP opposes the use of blockchain as ISP is a centralized platform.

In [28], a crowdsourcing framework for energy generation and trading is proposed assisted by blockchain. The work designs an incentive mechanism for traditional and distributed generation and distribution models in a way to meet the market equilibrium. In the work, IBM Hyperledger is used to enroll users before allowing them to query and invoke the smart contract functions. In addition, once the system operator determines the market equilibrium, smart contracts are created to record the transactions and reward the users accordingly.

In [29], a blockchain-based crowdsensing framework was proposed that tackles the location

privacy of workers in the system. The proposed framework protects workers identities and locations from being exposed by allowing them to submit to tasks either in the public or private blockchain by specifying the working region instead of their exact locations. In addition, workers' can spread their interactions over multiple private blockchains. Then, as a single user is not allowed to have access to all the private chains, the identity of the workers cannot be inferred by tracking their interactions. The synchronization between the private and public chains is managed by an intermediary agent.

3.4. Discussion

Table 1 presents a comparison of the surveyed works based on the shown metrics. Most of the existing distributed and blockchain-based works register users off-Chain through a trusted centralized entity. This limits the openness of the framework and retains the centralized framework limitations in terms of breaching users' privacy.

In the case of reputation, some of the existing works consider workers' reputation as an indication for their reliability, which is used to identify misbehaving workers when allocating tasks. However, all the existing works neglect the importance of requesters' reputation in motivating workers to accept tasks that belong to a given requester. Hence, the frameworks are considered unfair for workers and biased towards requesters as they do not give the workers enough information when tasks from multiple requesters are available. The only penalization, introduced by [24], for misbehaving requesters is the distribution of their deposits, which still does not allow workers to identify their consequent tasks.

Considering the allocation of tasks, surveyed works vary in executing the allocation on-Chain or off-Chain. Performing the allocation off-Chain allows using computationally expensive optimization techniques. However, off-Chain execution does not allow transparent verification by the framework members. Alternatively, works using transparent on-Chain allocation mechanisms use greedy mechanisms for their computational efficiency, hence cost efficiency. Yet, some of the works do not present the cost analysis of their proposed mechanism, which is of great importance.

While distributed efforts vary in deploying evaluation functions for workers' submission, blockchain-based efforts all consider evaluating workers' submissions for quality computation. Surveyed works differ in the source of the evaluation function where the majority of the efforts that conduct the evaluation, acquire the function from requesters. This opens the possibility for requesters to bias the function to reduce the quality of workers' submissions and consequently their payments, or their reputations. Moreover, the execution of the evaluation function is mostly done off-Chain by the requesters or central platform making it opaque. User privacy implied by their identity privacy and personal information is maintained in blockchain-based efforts through the pseudonym addresses used for on-Chain interactions.

The above limitations are considered when formulating our proposed *SenseChain* framework described in the following section. *SenseChain* relies on Ethereum blockchain as an infrastructure for users to register on-Chain. The framework maintains requesters' and workers' reputations to provide dual feedback on their reliability. Furthermore, it embeds the task allocation and solution evaluation mechanisms within the deployed smart contracts for on-Chain transparent execution.

Table 1: Blockchain-based Crowdsourcing: OF= Off-Chain, ON= On-Chain, hybrid= centralized and distributed, SC= Smart Contract, GT= Ground Truth, R= Requester, W= Worker

Paper	Platform	Registration	Reputation		Allocation	Evaluation Function		User Privacy	ON SC
			R	W		Source	Execution		
[14]	Hybrid	OF	No	Yes	OF	No	-	Yes	NA
[15]	Hybrid	OF	No	No	OF	Platform	OF	No	NA
[16]	Hybrid	OF	No	No	OF	No	-	No	NA
[17]	Distributed	OF	No	No	OF	No	-	No	NA
[23]	Blockchain	OF	No	Yes	OF	R	OF	Bitcoin Addresses	No
[24]	Blockchain	ON	No	Yes	ON	R	R & SC	Ethereum Addresses	Yes
[25]	Blockchain	OF	No	No	OF	R	R	Certificates	Yes
[26]	Blockchain	OF	No	No	OF	R	Miners	K-anonymity	No
[27]	Blockchain	OF	No	Yes	ON	GT	ISP	Temporary accounts	Yes
[28]	Blockchain	ON	No	No	OF	No	-	Fabric CA	Yes
[29]	Blockchain	ON	No	No	OF	No	-	Ethereum addresses, private blockchains	Yes
Proposed Framework	*Blockchain*	*ON*	*Yes*	*Yes*	*ON*	*ON*	*ON*	*Ethereum Addresses*	*Yes*

4. SenseChain Framework

In this section, the proposed *SenseChain* framework is described. First, the overview and assumption of the framework are discussed. Second, the designed smart contracts used in the framework are presented. Third and last, the process and its functions are described.

4.1. Overview and Assumptions

The proposed *SenseChain* framework leverages the existing 'multiple requesters with multiple workers' crowdsensing models, by moving from the centralized to decentralized blockchain-based models. *SenseChain* allows requesters and workers to interact and complete tasks without relying on a trusted centralized platform. Their interactions are governed by smart contracts running on the trusted Ethereum blockchain. Users interact with the framework by issuing transactions to Ethereum, which are executed by the miners in the network and verified by keccak256 hashing algorithm. The proposed framework is concerned with mitigating misbehaving participants, which aim to harm the crowdsensing activity being users or the platform. Fig. 2 shows the main components of the framework. Each of the components, users and smart contracts, has an Ethereum address to interact with other components in the framework knowing that all interactions are logged on the chain.

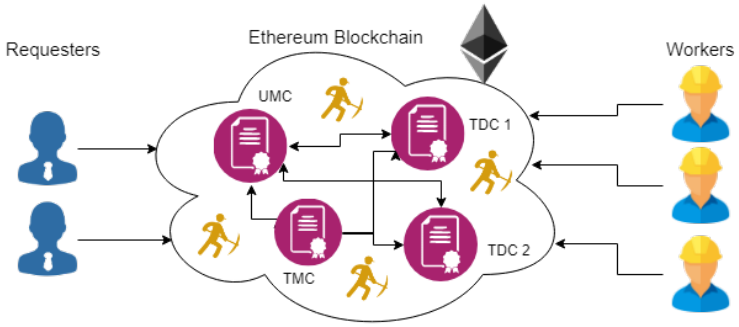


Figure 2: *SenseChain* Overall Model

The roles of these components are summarized below:

- **Requesters:** A requester registered in UMC is capable of publishing task requests to workers. To create a corresponding TDC for a task, the requester sends a task request to TMC along along with the budget it is willing to pay for workers once it is completed.
- **Workers:** A worker registered in UMC is able to participate in the framework and query TMC for available relevant tasks' TDCs. A worker can send reservation requests to tasks' TDCs. Then, if a slot reservation is accepted, the worker is expected to submit a solution and subsequently awarded in terms of payment and reputation accordingly.

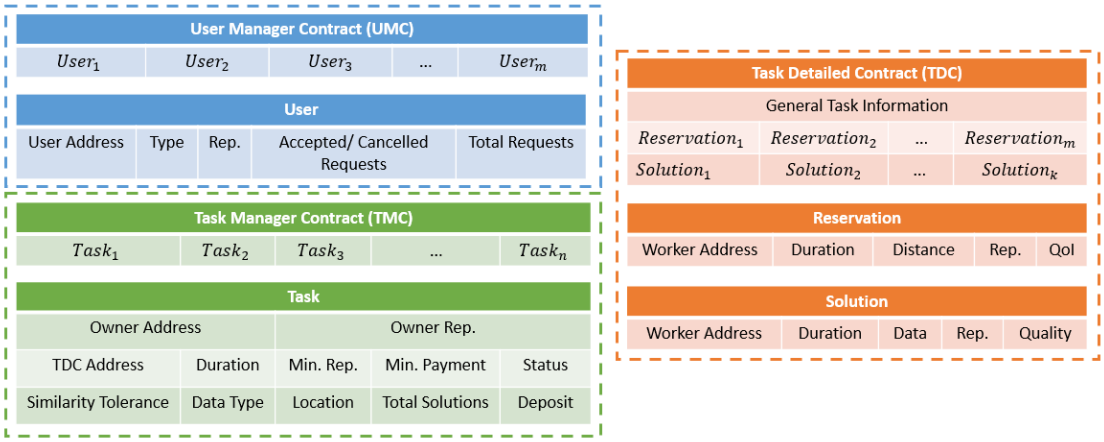


Figure 3: *SenseChain* Smart Contract

4.2. Smart Contracts

The **User Manager Contract (UMC)** shown in Fig. 3 maintains a list of users, requesters and workers, registered in the framework along with their information as *User* objects. A user is identified through a unique *Ethereum Address* to maintain users' privacy by not requesting further information. *Type* identifies the user's type, i.e. requester or worker, which defines his/her role in the framework. The *Reputation (Rep)* presents a user's reliability given his/her previous interactions. *Rep* is set to an initial value by the contract once a user is registered and is updated according to his type and interactions. Lastly, request statistics, i.e. *Accepted/ Cancelled Requests* and *Total Requests*, are used for reputation calculation. For a requester, *Cancelled Requests* implies the number of cancelled requested tasks while *Total Requests* implies the total number of created requests. On the other hand, for a worker, *Accepted Requests* represents the number of the successfully completed tasks while *Total Requests* represents the total number of reserved tasks.

UMC includes a *addUser()* function which allows user to register with the framework smart contract. The *updateStatistics()* and *updateReputation()* functions are for TDC to update a user's information.

The **Task Manager Contract (TMC)** is a directory for pending tasks saved in a list of *Task* objects. A *Task* object includes the general information about a task as shown in Fig. 3. The *Owner Address* and *Rep* are included for workers to differentiate requesters according to their reputation. The *TDC Address* holds the address of the task's TDC, which workers will interact with. The *Status* of a task can be: pending, completed or cancelled. The *Data Type*, *Duration*, *Location*, *Min. Rep.* attributes are criteria for workers to determine their eligibility for a task. The *Deposit* presents the total budget a requester is willing to pay for the task while *Min. payment* is the minimum guaranteed payment a worker gets for completing the task. The *Min. payment* is calculated using the *Total Solutions* required for the task by the requester. The *Similarity Tolerance* is a metric utilized in evaluating solutions implying the maximum similarity difference between accepted solutions.

TMC includes *createTask()* function which requesters invoke to publish their task and

create the corresponding TDC by passing the task attributes.*updateStatus()* and *reduceSolutions()* are included for internal update of task information based on users' interactions.

The **Task Detailed Contract (TDC)**, shown in Fig. 3, is created for each task request sent to TMC. A task's TDC includes its general information in a *Task* object along with workers' *Reservation* and *Solution* objects whose attributes are shown in Fig. 3. Different functions exist in TDC such as *addReservation()*, *evaluateReservation()*, *addSolution()*, and *evaluateSolutions()*. *addReservation()* takes users' reservation attributes and creates a *Reservation* object while *evaluateReservations()* evaluates submitted reservations and creates a *Solution* object for each accepted reservation. *addSolution()* takes the submitted solution by a user and updates the *Solution* object with the submitted value. *evaluateSolutions()*, which runs when all solutions have been submitted or the task expires, calculates the quality of submitted solutions to determine users' payments and reputation updates according to a built-in evaluation function. TDC's main advantage to crowdsensing is its ability to perform autonomous unbiased evaluation, payment sharing and reputation update.

4.3. Process and Functions

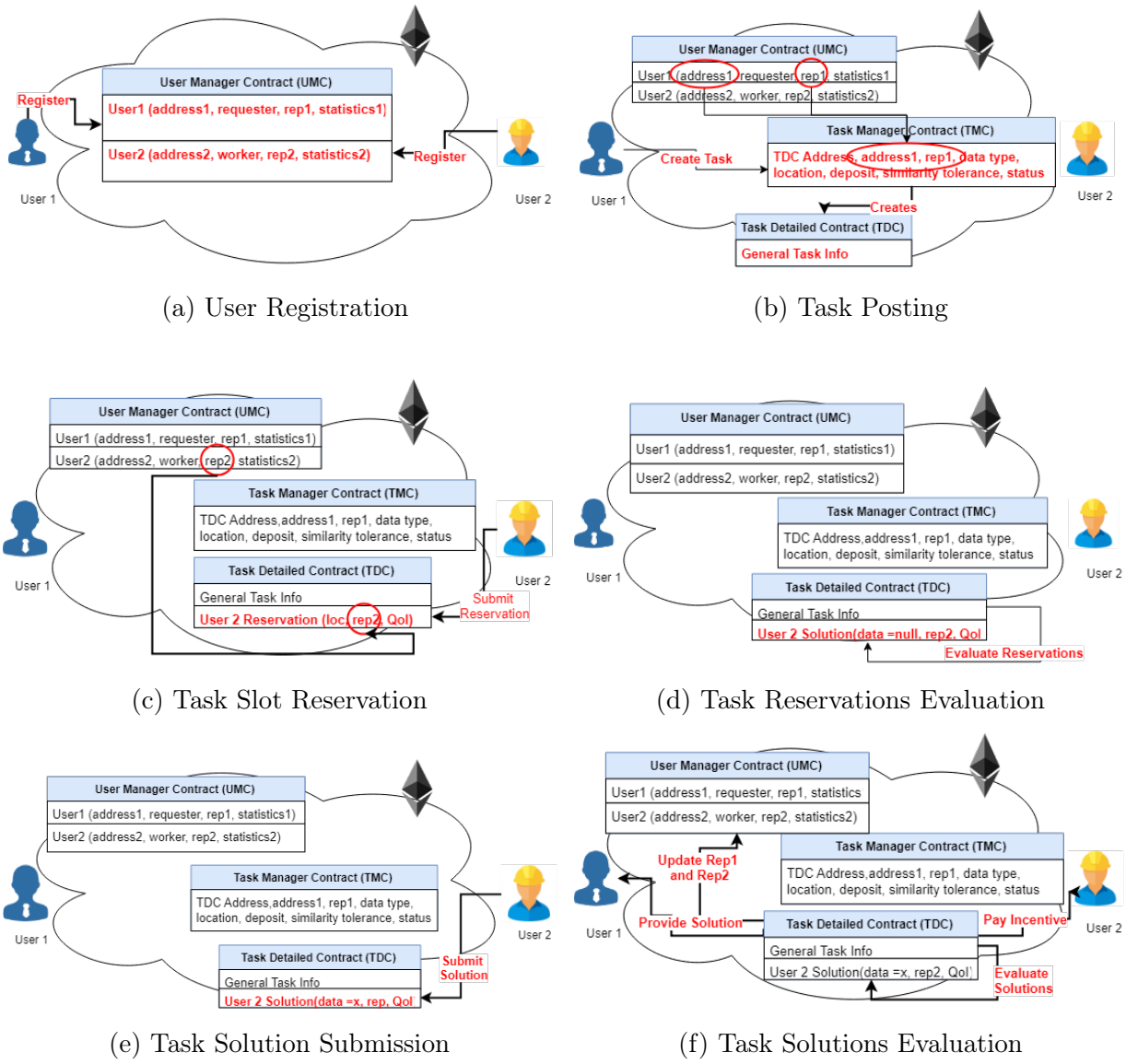
The process of *SenseChain* framework is illustrated in Fig. 4. Ethereum blockchain and the deployed smart contract objects, UMC, TMC, and TDC, are abstracted and presented as needed along with the information saved in them. The interactions of the users, requesters and workers, are shown with the updated information highlighted in red. Assuming that UMC and TMC are pre-deployed, the rest of the steps are detailed as follows.

4.3.1. User Registration

Requesters (e.g. user 1) and workers (e.g. user 2) register to the global UMC once to be able to participate in the framework whether to publish tasks or submit to pending tasks. They invoke the UMC's *addUser()* function and pass their Ethereum addresses and registration type. The function initializes the reputation and statistics values and adds their entries to UMC as shown in Fig. 4a. A user is assumed to have a single Ethereum address associated to their reputation to present previous interactions with the framework. The user's reputation can only be changed by the smart contracts through their internal calls and is read from UMC when needed. Hence, the reputation cannot be manipulated by users since it is acquired and updated by the trusted miners in the blockchain.

4.3.2. Task Posting

A requester, such as user 1, can post a task by passing its attributes presented in Fig. 3 and the task's budget to the *addTask()* function in the global TMC. *addTask()* checks the requester's entry in UMC to retrieve the reputation if registered or revert the request if the user is not registered. For a valid requester, TMC creates a TDC with a unique Ethereum address for the remaining stages of the task, which it transfers the task's budget to. The TDC's Ethereum address and task attributes are encapsulated into a *Task* object which is appended to the list of pending tasks in TMC as shown in Fig. 4b.



(a) User Registration

(b) Task Posting

(c) Task Slot Reservation

(d) Task Reservations Evaluation

(e) Task Solution Submission

(f) Task Solutions Evaluation

Figure 4: *SenseChain* Process with the interactions of the workers, requesters, and the deployed smart contracts. UMC and TMC are pre-deployed before the steps are permitted. Actions, changes to the state and the link between metrics are highlighted in red at the different steps.

4.3.3. Task Slot Reservation

A worker interested in performing a task, e.g. user 2, is required to reserve a solution slot for a given task. Reservations is a common mechanism used in crowdsensing/ sourcing to utilize the resources of a worker as is the case in Uber [3]. A worker would not start performing a task and consume resources before the reservation is accepted. This is of concern when many workers are involved in the framework while a limited number of submissions is requested. Without the reservation mechanism, workers can complete the task and submit their contributions while the task has already received the required number of submissions.

Thus, a reserved slot guarantees that a worker's solution will be evaluated if submitted before the end of the task duration.

First, a worker queries the list of pending tasks in TMC to identify tasks eligible tasks for based on their requirements. Next, the worker locally calculates the Quality of Information (QoI) metric of these tasks. The QoI is used as it is a standard metric for crowdsensing, which integrates multiple independent metrics; reputation, time, and distance. In this work, the worker determines the highest QoI task and submits a reservation request by invoking the *addReservation()* function of its TDC, since it is assumed to be the task most likely to accept it. TDC acquires the reservation attributes from the worker's request (*location and duration*) and UMC (*Rep*) to recalculate the worker's QoI construct a *Reservation* to be added to the *Reservations* list, as shown in Fig. 4c. The worker's QoI for a task is calculated according to Eq. 1:

$$QoI_{w,t} = \frac{Rep_w}{ST_{w,t} \times D_{w,t}} \tag{1}$$

where w is a worker, t is a task, $QoI_{w,t}$ is the QoI of w for t, Rep_w is the reputation of w, $ST_{w,t}$ is the maximum time required by w to submit the data to t, and $D_{w,t}$ is the distance between w and t.

A misbehaving worker can choose to submit a lower reservation time than needed for a higher QoI value. However, the worker risks being penalized if the task is not completed or completed with a low-quality solution. Similarly, forging the distance affects the quality of the submitted solution, especially that other workers might be selected within a shorter distance from the task with more accurate submissions.

4.3.4. Task Reservations Evaluation

Once the reservation interval expires, the *evaluateReservations()* function in a task's TDC is invoked as shown in Fig. 4d. It evaluates the submitted reservations and accepts the set of workers with maximum QoI based on the number of workers required by the requester. For accepted workers, a *Solution* object is created and added to the *Solutions* list for the reserved duration. The reservation is used to identify and penalize workers who reserve but do not submit solutions by the end of their reservation period by reducing their reputation. It is worth noting that a slot is reserved using the worker's address. Consequently, a worker cannot reserve multiple slots from a single account. However, as workers' identities are hidden, a worker can create multiple ones and submit from them instead of stealing the credentials of other registered workers. A fee is endured by each address independently for the function's execution, which would reduce the probability workers submitting from multiple addresses.

4.3.5. Task Solution Submission

Workers with accepted slots are notified, where they can send their solutions anytime before their reservation slot expires. In Fig. 4e, after user 2's reservation is accepted, the user travels to do the task and is expected to submit the solution to TDC before the end of the reservation interval. Once the solution is available, user 2 invokes the *addSolution()* function in the task's TDC to submit the collected solution by updating the data field in its

Solution object. If a slot expires, *addSolution()* function is automatically invoked to release the worker's slot for other workers to reserve and submit their solutions.

4.3.6. Task Solutions Evaluation

Once a task is completed, the submitted solutions are evaluated to determine their solution qualities as shown in Fig. 4f. The solution qualities are important as they influence workers' payments and reputations. The *evaluateSolutions()* function responsible for the evaluation is invoked by two triggers events and time. The event trigger happens when a worker invokes the *addSolution()* function which checks the number of remaining solutions, if none is remaining the *evaluateSolutions()* function is called. The second trigger is time-based, where the *evaluateSolutions()* is scheduled for execution after the task expires. This overcomes the deadlock situation where the task is waiting for a submission from a worker intentionally or unintentionally missing the reservation slot.

Solution Quality: In *evaluateSolutions()*, the quality of submitted solutions is determined As inspired from [39], the quality of a solution is proportional to its similarity to other solutions submitted to the task, given honest workers will have similar submissions. The solution quality is calculated according to Eq. 2. This assumes a sensing task asks for quantifiable values collected from multiple workers. The equation can be changed based on the nature of the readings.

$$SolutionQuality_w = \frac{\sum_{j \in C/\{w\}} f(w,j)}{size(C) - 1} \tag{2}$$

$$f(w,j) = \begin{cases} \frac{100}{|val(w) - val(j)|} & : val(w) \neq val(j) \\ 100 & : val(w) = val(j) \end{cases} \tag{3}$$

where w is worker, $SolutionQuality_w$ is the quality of w's solution, $val(w)$ presents the submitted data value by w, and C is the set of solutions submitted, assuming a task requires more than one submission. It is worth noting that as Solidity takes only integer values, the minimum difference between two unequal readings is one.

The solution quality is important to determine each worker's corresponding payment and reputation update. The autonomous nature and transparency of the quality calculation makes it unbiased to intervention of requesters, workers and the platform, which is not guaranteed in the case of centralized frameworks. In addition, the evaluation runs even when a requester attempts to cancel the task. That ensure workers who are executing the task are given their payments as the budget is deposited earlier by the requester.

Worker Payment: After the solution quality evaluation, workers payments are determined to be paid directly by the TDC in terms of Ether from the deposited budget as Fig. 4f shows. The payment of each worker is divided into two parts: minimum payment and quality-based payment. The minimum payment, calculated before the task starts, is the least payment a worker gets for completing the task successfully. A fraction of the deposit, α, is used to determine the *MinPayment* equally divided on the maximum number of workers

expected to submit as indicated in Eq. 4:

$$MinPayment = \frac{\alpha \times Deposit}{n} \tag{4}$$

where *Deposit* is the budget of the task, α is the fraction of the budget used for the minimum payment, and n is the maximum number of required workers for the task.

The quality-based payment is an additional payment rewarded given a worker's solution quality. First, the *RemainingDeposit* for the quality-based payment is calculated after the task is completed and submitted workers are determined, as shown in Eq. 5:

$$RemainingDeposit = Deposit - (MinPayment \times submitted) \tag{5}$$

where *submitted* is the number of submitted solutions by the end of the task.

Second, the quality-based payment for a worker is calculated using the remaining deposit and the quality of his solution, as shown in Eq. 6:

$$QualityBasedPayment_w = RemainingDeposit \times \frac{SolutionQuality_w}{\sum_{j \in C} SolutionQuality_j} \tag{6}$$

where w is a worker, *RemainingDeposit* is calculated according to Eq. 5, and $SolutionQuality_w$ is calculated according to Eq. 2.

Third and last, the payment of a worker, such as user 2, is given according to Eq. 7:

$$Payment_w = MinPayment + QualityBasedPayment_w \tag{7}$$

where w is a worker, $MinPayment$ is calculated using Eq. 4, and $QualityBasedPayment_w$ is calculated using Eq. 6.

Workers with solution qualities outside the task's *similarity tolerance* metric are not paid as their solutions deviate from the other submission with a value greater than the tolerance constrained by the requester. The impact of misbehaving platform and requesters interested in reducing the quality of workers' solutions for lower payment is eliminated by unbiased evaluation. In addition, the payment is calculated and transferred to each worker by TDC without any intervention from the requester or an untrusted platform. This guarantees workers receive their entitled payments based on their submissions. If a task expires without receiving reservations, the deposit is returned to its requester.

Reputation Update: The *evaluateSolutions()* function invokes the UMC's *updateReputation()* function once payments are completed to update the reputations of the requester and involved workers. The updated is done according to the final status of the task and workers' solution qualities. A reputation mechanism that motivates conforming users while penalizing misbehaving ones is required. The reputation of a requester presents the number of tasks they cancelled, while for a worker it denotes the submitted solutions' quality. While the reputation can rely on instantaneous interactions, historical interactions are typically considered for more reliability and consistency over time. The proposed reputation mechanism is inspired from the Transmission Control Protocol timeout period calculation, which takes int consideration

both the estimated and sampled timeout values [40]. Other mechanisms that meet the requirements of our platform do exist and are expected to provide the same trends as long as they penalize misbehavior and reward trustful submissions. For a requester, if the final status of the task is cancelled, the *CancelledRequests (CR)* and *TotalRequests (TR)* attributes are incremented, while only the TR attribute is incremented otherwise. Then, the requester's reputation is calculated according to Eq. 8.

$$Rep_r \leftarrow 100 \times ((\beta \times \frac{Rep_r}{100}) + (1 - \beta)(1 - \frac{CR}{TR})) \tag{8}$$

where r is a requester, β is a weight factor, Rep_r is the reputation of r, CR is the number of cancelled requests, and TR is the total number of requests. Rep_r is divided by 100 to normalize it and have both components equivalent.

Similarly, when a worker's submission is accepted by the evaluation, the *AcceptedRequests (AR)* and TR attributes are incremented, while only the TR attribute is incremented otherwise. The worker's reputation is calculated according to Eq. 9.

$$Rep_w \leftarrow 100 \times ((\beta \times \frac{Rep_w}{100}) + (1 - \beta)(\frac{AR}{TR})) \tag{9}$$

where w is a worker, Rep_w is the reputation of w, AR is the number of accepted requests, and TR is the number of total requests.

The reputations of requesters and workers are updated by the end of each task to improve the selection in future tasks. For requesters, their reputation is linked to their cancellation rate where it decreases with the increase of the number of cancelled tasks which workers have reserved slots in. In contrast, for workers, their reputation is linked to their successfully completed tasks where it increases in their increase.

Overall, the proposed framework tackles existing challenges in crowdsensing frameworks by its features as summarized below:

1. Trust is maintained by building the crowdsensing framework on top of Ethereum blockchain.
2. The selection of user is done in a transparent and unbiased manner using smart contract, hence guaranteeing its reliability.
3. Users' payments are guaranteed by making the payments from smart contracts. The contracts require the deposit from requesters before the task starts and autonomously shares the payment with workers once completed.
4. In addition to misbehaving workers, misbehaving requesters are penalized by reducing their reputation for workers to identify them in future tasks.

5. Security Analysis

In the proposed SenseChain framework, security is inherent from the underlying Ethereum blockchain and from the deployed mechanisms. Three main security problems that centralized crowdsensing frameworks suffer from are considered:

Identity Privacy: Centralized crowdsensing frameworks require users to register their personal information to benefit from the crowdsensing service. A trusted third party is typically responsible for maintaining this information. However, trusted third parties can breach users' privacy and compromise their identities for their own benefit.

In our proposed framework, users still need to register to benefit from a crowdsensing service. However, workers and requesters use their generated Ethereum addresses to interact with the framework. As the addresses are pseudonym public keys, the users' identities are protected, and hence cannot be exploited and exposed.

Trusted Execution: In a typical centralized crowdsensing platform, the different mechanisms such as submission evaluation and payment computation are executed by the platform or the requesters in a hidden process. This execution can be manipulated to bias the outcome for the benefit of either the requester or the platform. For example, the requester can manipulate the quality of the received submissions to reduce the workers' payments. Similarly, the platform can underpay the workers after the fulfillment of a given sensing task.

SenseChain tackles this challenge by embedding the evaluation function in the smart contract (TDC). When the task finishes, the evaluation is executed and consensus on its outcome from independent nodes is reached, preventing the evaluation from being biased. Additionally, the payments are calculated by the smart contracts and securely dispatched to workers from the deposited task budget.

Dual Feedback: All the surveyed crowdsensing frameworks incorporate the reputation of workers to provide requesters with feedback on workers' reliability. However, the inverse mechanism is not implemented. Providing workers with feedback about task requesters can motivate them to accept a requester's task. This is even more important when many sensing tasks from different requesters are available. Using the requesters' reputations/feedback, workers can identify misbehaving requesters, i.e. requesters who issue redundant tasks to consume workers' resources and overload the platform, requesters who frequently cancel tasks after allocation of workers, etc.

In our proposed framework, the reputations of both the requesters and workers are provided to implement a dual feedback mechanism. Hence, both requesters and workers have prior knowledge of the reliability of each other before committing to a task or accepting a worker.

6. Performance Evaluation

This section describes the performance evaluation conducted to demonstrate the feasibility of the proposed *SenseChain* framework while comparing its performance to the greedy centralized framework proposed in [6]. First, the evaluation setup is described. Second, the performance results are illustrated to show the comparability of the proposed framework for a fixed set of workers. Third, the evaluation is done for a varying number of workers.

6.1. Evaluation Setup

To implement *SenseChain*, Solidity (version 0.4.21) [30], the programming language for Ethereum, was used to create and compile the smart contracts constituting the framework.

Web3j [41], a library for java applications on the Ethereum network, was used for interacting with the contracts along with a locally constructed website. A local test blockchain was created with multiple accounts for the users and an initial wallet of 100 Ether in each account. The number of workers was fixed for part of the evaluation. Where mentioned, the number of workers was increased to study the performance when varying the number of workers.

A publicly available radiation dataset [42] is used to measure the performance of the model. The dataset includes radiation readings collected at different intervals of time from approximately 500 sensors (workers) at different locations. These readings are used to assess the quality of the selected workers and their submitted data. As reputation is not included in the dataset, it is initialized to 50 for all workers and is updated accordingly. Table 2 presents the attributes of workers from the dataset used in the evaluation.

Table 2: Dataset Attributes

Workers	$\approx$ 500 (490)
Solution Submission Time (interval)	[1,24]
Latitude and Longitude	[35,44] and [136,142]
Data Collected	Radiation Level (μSV)
Initial Reputation	50

Table 3 presents the parameters used to generate tasks. The task budget is varied for the different tasks as normally done by different requesters. The minimum budget is set to \$0.1 as in [2]. For the duration, tasks are assumed to span over 1 hour slots for a maximum of 24 hours. The location of a task is randomly generated within the area of the workers to ensure at least a single worker will be selected for the task. The number of total solutions is randomly selected between 5 and 7 submissions to be able to detect difference in received task's submissions. As the similarity tolerance restricts the acceptance of lower quality solutions, a random value within band of 10% is considered during the evaluation, knowing that submissions are taken as integer values. The cancellation round presents the number of rounds the requester waits before cancelling its task. It is selected between [1,30] where a bigger value reduces the probability a task will be cancelled as it might be completed before that. α is chosen as 0.5 for equal budget division between the minimum payment and the quality-based payment. Lastly, β was selected as 0.75 to strengthen the contribution of historical interactions of a user compared to the current interactions.

Multiple iterations were run to collect the results with each iteration being a complete run of the framework rather than a software simulation (e.g. Matlab). For each iteration, an independent set of tasks was generated in random locations within the workers' area, where the number of tasks is incremented when increasing the number of workers. Each point in the results is acquired from the average of five iterations.

6.2. Benchmark

To be able to measure the performance of the proposed model, it is benchmarked against a centralized greedy selection framework for crowdsensing [6]. This comparison allows for the

Table 3: Task Generation Parameters (Per Task Attributes)

Number of Tasks	30 per 100 workers
Tasks per Requester	6
Data Type	Radiation
Task Budget	[\$0.1,\$4] $\approx$ [0.5,25] Finny
Duration (in hours)	[0,24]
Location (Latitude, Longitude)	[[35,44],[136,142]]
Total Solutions	[5,7]
Similarity Tolerance	[5,15]
Cancellation Round	[1,30]
α (Fraction of budget for minimum payment)	0.5
β (Weight factor for reputation contribution)	0.75

quantification of the difference in performance between an optimized centralized framework and *SenseChain*. In the greedy model, all tasks are assumed to co-exist in the framework when the workers' selection is conducted. For each task and each user in the framework, the minimum reputation required by the task is checked to determine the eligibility of the worker. Second, the distance between workers and tasks, as well as the QoI of workers are calculated. Third, the best QoI (task, worker) tuple found is taken as the accepted selection and the total number of required workers by the task is reduced. These steps are repeated until the best set of workers is assigned, in a greedy manner, to each sensing task.

SenseChain is not compared to surveyed distributed models due to the fundamental difference in the desired objectives. The latter models run the framework in a distributed manner for task selection, analysis of data, or group formation without considering the trust of the framework. On the other hand, the former deploys the crowdsensing framework on a decentralized infrastructure to run the centralized function in a trusted manner.

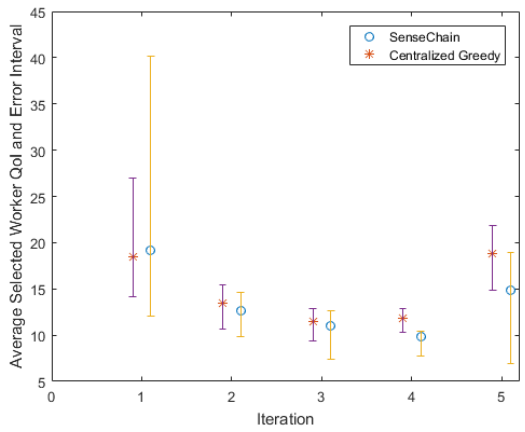
6.3. Evaluation Results

For the first part of the evaluation results, *SenseChain* is compared to the centralized framework while fixing the number of workers to 100. That permits understanding the performance when the workers are fixed while the tasks are varied in the consecutive iterations. For the second part of the evaluation, the number of workers and tasks is varied to understand how the framework scales with additional workers. The evaluation was limited by the number of workers in the selected dataset.

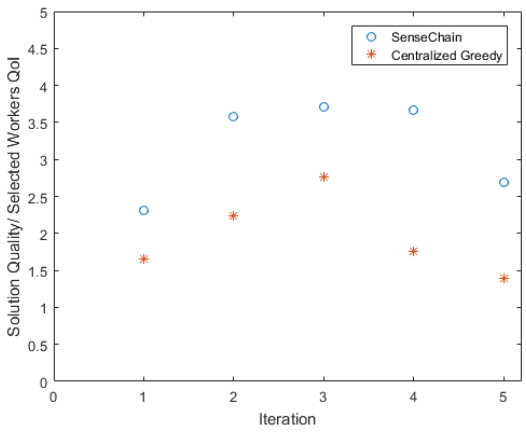
6.3.1. Fixed Number of Workers

Average Selected Workers' QoI: Fig. 5a presents the average QoI of selected workers based on Eq. 1. The QoI of a task is measured as the average QoI of workers selected to perform it as Eq. 10 shows.

$$AverageSelectedWorkersQoI_t = \frac{\sum_{m \in S_t} QoI_m}{|S_t|} \tag{10}$$



(a) Average Selected Workers' QoI



(b) Solution Quality given Selected Workers' QoI

where t is the task, and S_t is the set of selected workers for t.

The results shown present the average, minimum and maximum QoI of selected workers for tasks. The centralized model results in a higher average QoI than *SenseChain* as a consequence of the optimized allocation of workers to tasks in all iterations. However, it is notable that the difference in the average QoI is relatively small in most iterations which makes *SenseChain* a viable solution. The drop in QoI is a natural consequence of the distributed nature of workers' selection. Hence, other metrics are measured to have a better assessment of the model.

The similarity of the selected workers in both frameworks, centralized and *SenseChain*, is measured to present the impact of the distribution on the selected workers. The similarity between selected workers was found to be 40.88%, which reflects that both models select different workers for the sensing tasks. The difference is as a consequence of the nature of the selection where the centralized platform has full prior knowledge about the tasks and the workers. In contrast, *SenseChain* allows each task to select workers independently without knowledge of the other running sensing tasks.

Solution Quality: While the QoI of selected workers is important to measure, it is a pre-selection metric. Meaning, it reflects how good are the selected workers based on parameters such as their reputation, distance from the task, etc. However, QoI as defined, cannot predict how good is the workers' submitted data. Hence, the quality of submitted solutions, calculated using Eq. 2, is studied. Table 4 shows that *SenseChain* results in a solution quality of 41.21 while the centralized model results in 27.79 making *SenseChain* better by 48.24%. The better performance in *SenseChain* implies that although selected workers in the centralized are better in terms of QoI, it is not necessarily the case for the solution quality as shown in Fig. 5b. This is because the solution depends on the shared data by the worker and not his local (distance, duration) or historical (*Rep*) information.

Solution Quality given QoI: Fig. 5b presents the ratio between submitted solutions' quality and the QoI of selected workers to perform the task. This metric links the two previous metrics to understand the relation between the post-selection (solution quality)

Table 4: Performance Comparison

Performance criteria	Centralized	SenseChain	Difference
Selected Workers' QoI	11.26	9.48	-15.78 %
Submitted Solution Quality	27.79	41.21	48.24 %
Time Cost	11	11.1	-0.9%

and pre-selection (QoI) performances. A higher ratio imply that the difference between the two metrics is big making the lower QoI acceptable. *SenseChain* clearly outperforms the centralized model in the different iterations with at least 43%. That implies that the proposed framework provides a better quality of solutions, compared to the centralized model, even with its comparable QoI of selected workers. This proves that despite the optimization of the worker selection in the centralized framework based on workers' parameters, the selected set might not be the best in terms of collected data. In turn, the deployment of the proposed model as a potential solution for sensing frameworks is encouraged.

Time Cost: Table 4 illustrates the average time cost for completing the task based on the selected set of workers. For instance, a time cost of 11 implies that given the set of selected workers to perform the task, the longest interval a worker requires to submit its solution is 11. In this scenario, both centralized and *SenseChain* shows a minor difference (0.9%) in the time cost of tasks, which supports the applicability of the proposed model. Moreover, it demonstrates that even with the difference in selected workers between the frameworks, a similar result can be found in terms of time cost.

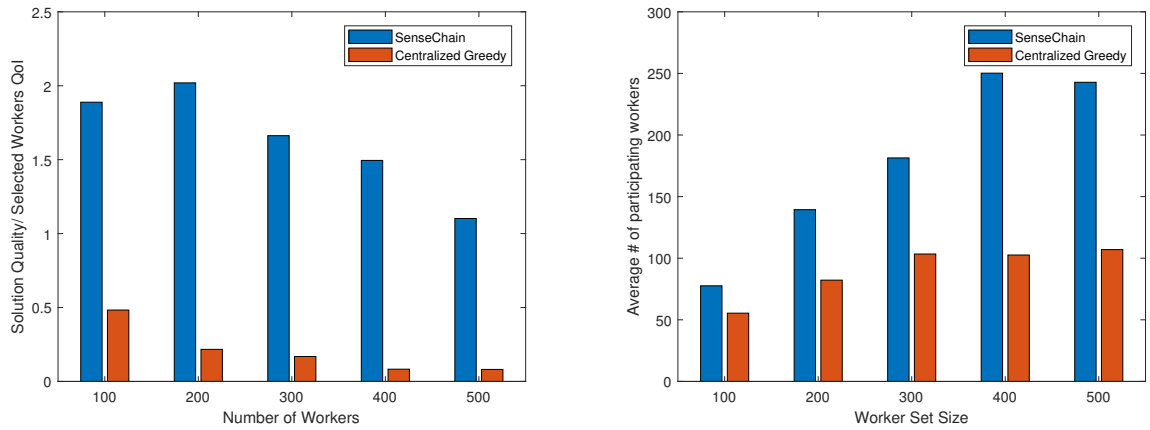
6.3.2. Varied Number of Workers

The number of workers and tasks was extended in the to evaluate the framework performance when more workers are involved. The evaluation is conducted by increasing the number of workers progressively from 100 to 500 (dataset size) with increments of 100. The number of tasks is incremented according to Table 3.

Solution Quality given QoI: Fig. 6a presents the ratio between submitted solution qualities over the expected QoI. As the solution quality given QoI metric is representative of both independent metrics, it is the evaluated metric in this scenario. It can be seen that *SenseChain* results in a much better ratio than the centralized greedy framework even at the different number of workers evaluated. This is important as a task is not only concerned with the QoI of the selected workers but the resulting solution qualities of these workers. The metric drops when increasing the number of workers as both metrics.

Number of Selected Workers: Fig. 6b shows the number of selected workers to perform tasks for different numbers of workers. It can be clearly seen that *SenseChain* selects a higher number of workers compared to the centralized greedy framework for the different considered sizes. While the increase is evident in *SenseChain*, the number of workers seems to converge in the centralized greedy when more workers and tasks are introduced.

That is due to the selection of the best workers in a centralized optimized manner where these workers will do all the tasks assuming they don't have a limit on the number of tasks to perform. On the other hand, *SenseChain* engages more workers with the framework because



(a) Solution Quality given Selected Workers' QoI (b) Number of Selected (Participating) Workers

of its distributed nature. The scalability of blockchain is further discussed in section 7 as it impacts the performance of the framework and its feasibility.

7. Ethereum Cost Analysis and Scalability

The cost of deploying the framework is measured to validate its feasibility in terms of the required cost. It is measured on two folds: 1) cost of each functionality in the framework, and 2) cost endured by each user in the framework. Three units are used to present the cost: 1) gas consumption, 2) Ether and 3) USD, where each is dependent on the other. The gas consumption is directly related to the size of the deployed Smart Contracts and the size of information saved into them. The Ether cost is dependent on the gas price required for miners to execute the transactions of the functions in the model, where a higher price entails faster execution of transactions. In this work, a gas price of 5.1 Gwei is used for a maximum 2 minutes confirmation time (0.57 minutes in average), which is the standard gas price in the literature [43]. The gas price impacts the time needed to execute a functionality where a higher price results in faster execution. Hence, the time cost of executing the framework functions could not be presented due to variations in what users are willing to pay. Lastly, the conversion between Ether and USD is governed by the currency trading price [44]. Hence, cost in terms of USD can differ in time.

7.1. Functionality Costs

Table. 5 presents the cost of each functionality in the framework in three different units. *create UMC and TMC* are one-time functions that are called to create the framework and their cost is less than $1 combined. This implies that the cost of deploying the framework is minimal with no further cost required to maintain it. The function *addUser()* in UMC, a one-time function called by users, costs less than $0.05 on average where no other fees are paid by users to maintain their registrations. For *createTask()* called by a requester, the cost is $1.1704 which includes the cost of creating the TDC and adding the task's information to

Table 5: Cost of Each Framework Function

Function	*createUMC*	*createTMC*	*addUser*	*createTask*
Gas	463739	1312667	80294	2239934
Cost(Ether)	1.16 E-03	3.28 E-03	2.01 E-04	5.6 E-03
Cost(USD)	0.2423	0.6859	0.0420	1.1704
Function	*addReservation*	*evaluateReservations*	*addSolution*	*evaluateSolutions*
Gas	125598	752208	56175.5	702808
Cost(Ether)	3.14 E-04	1.88 E-03	1.40 E-04	1.76 E-03
Cost(USD)	0.0656	0.3930	0.0293	0.3672

the list of pending tasks in TMC. As this function has the highest number of lines to execute, it has the highest cost compared to the other functions.

The *addReservation()* function costs less than $0.07 per reservation which is marginal compared to the payment a worker is expecting to collect from a task successfully completed. The function *evaluateReservations()*, which is run by the TDC, costs $0.393 on average to generate the set of accepted reservations while *addsolution()* costs an additional $0.0293 on average to submit their solutions. Lastly, TDC requires almost $0.3672 to *evaluateSolutions()* which entails calculating the quality of the submission, calculating the payment for workers and updating their reputations.

Overall, functions' collected costs are considerably small which motivates the use of the framework as a potential alternative to centralized frameworks. Moreover, these costs are based on the interactions of the users with the framework, meaning users only pay for what they use with no additional payments or charges as in centralized frameworks. The cost of centralized models is not presented due to the complexity in estimating its value and the variations in its calculations. For such models, the cost is typically dependent on the investment of the central management entity and the business model adopted.

7.2. User Costs

The cost per task endured by users (requester and worker) is measured to understand the cost according to a user's role. This is important as the distributed model allows users to pay fees according to their role, which efficiently reduces their payments. Table 6 presents the total cost of a requester and a worker when they are engaged in a complete cycle of a task from its creation to its submissions' evaluation.

Table 6: Framework Cost per User per Task

User	Requester	Worker
Total Gas Consumption	3694950	181773
Total Cost (Ether)	9.2373E-3	4.5439E-4
Total Cost (USD)	1.9305	0.0950

It is notable that the cost of the requester in this case is the highest; $1.9305 on average.

This is because a requester is the task publisher and pays for the execution of its operations; creating the task, evaluating reservations, and evaluating solutions. On the other hand, a worker endures the least cost in this model ($0.0950) on average which motivates workers to participate in the framework. These costs are independent of the number of workers and tasks deployed in the framework as they are measured for independent users. However, the costs can vary given the submitted data type by workers and the number of required submissions for a task, which will be studied in future work. Overall, both the costs of requesters and workers are considered acceptable due to their small values.

7.3. Blockchain Scalability

Scalability is one of the major challenges in blockchain specially with the increasing number of users and blocks added to it. The bottleneck in blockchain comes from the deployed consensus protocol. In the public blockchain 1.0, Proof of Work (PoW) [45] is used where it allows only a few transactions per second (tps) [46] making it highly unscalable. Proof of Stake (PoS) [47] or Proof of Assignment (PoA) [48] are other protocols which improve tps, hence the scalability of the blockchain. Ethereum blockchain is migrating to the PoS protocol by 2020 to improve its tps count from 20 tps [46] and its scalability [49]. Meanwhile, blockchain 3.0 is rising allows thousands of transactions to be processed in a second due to its consensus protocol (PoA) which efficiently adds transactions.

For our framework, the scalability of the framework is dependent on the scalability of the trusted blockchain infrastructure rather than the untrusted centralized infrastructure. While the scalability might be of concern, it's currently being handled by changing the consensus protocol and is expected to be resolved in the near future specially in blockchain 3.0 [49].

8. Conclusion

In this work, a blockchain-based crowdsensing framework for multiple requesters and multiple workers, called *SenseChain*, which overcomes the challenges of the centralized framework and requires reasonable cost, is presented. *SenseChain* utilizes Smart Contracts to replace the centralized platform by 1) registering users and maintaining their information reliably, 2) collecting and publishing tasks, 3) selecting workers in an unbiased manner, and 4) transparently evaluating solutions and sharing proportional payments with workers. The proposed framework provides a generic decentralized platform for both requesters and workers while allowing centralized-like functions in a trusted manner. This motivates the engagement of both workers and requesters and centralizes their resources to a trusted framework.

When comparing *SenseChain* to a centralized greedy selection, it is notable that the performance is similar in terms of the quality of selected workers, distance travelled, task completion duration, and submitted solution quality. In addition, the low deployment cost of the framework affirms the viability of blockchain-based crowdsensing frameworks.

Acknowledgment

This work was supported by the Khalifa University Internal Research Fund (KUIRF Level 2) under Project 8474000012.

References

[1] L. G. Jaimes, I. J. Vergara-Laurens, A. Raij, A survey of incentive techniques for mobile crowd sensing, IEEE Internet of Things Journal 2 (5) (2015) 370–380. doi:10.1109/JIOT.2015.2409151.

[2] Amazon Mechanical Turk (2018).
URL https://www.mturk.com/

[3] Uber (2018).
URL https://www.uber.com/ae/en/

[4] D. Difallah, E. Filatova, P. Ipeirotis, Demographics and Dynamics of Mechanical Turk Workers (August) (2018) 135–143. doi:10.1145/3159652.3159661.

[5] Companies using amazon mechanical turk (2019).
URL https://discovery.hgdata.com/product/amazon-mechanical-turk

[6] B. Guo, Y. Liu, W. Wu, Z. Yu, Q. Han, Activecrowd: A framework for optimized multitask allocation in mobile crowdsensing systems, IEEE Transactions on Human-Machine Systems 47 (3) (2017) 392–403. doi:10.1109/THMS.2016.2599489.

[7] M. Abououf, R. Mizouni, S. Singh, H. Otrok, A. Ouali, Multi-worker multi-task selection framework in mobile crowd sourcing, Journal of Network and Computer Applications.

[8] M. Abououf, S. Singh, H. Otrok, R. Mizouni, A. Ouali, Gale-shapley matching game selection—a framework for user satisfaction, IEEE Access 7 (2019) 3694–3703. doi:10.1109/ACCESS.2018.2888696.

[9] R. Estrada, R. Mizouni, H. Otrok, A. Ouali, J. Bentahar, A crowd-sensing framework for allocation of time-constrained and location-based tasks, IEEE Transactions on Services Computing (2018) 1–1doi:10.1109/TSC.2017.2725835.

[10] Y. Cheng, X. Li, Z. Li, S. Jiang, Y. Li, J. Jia, X. Jiang, Aircloud: A cloud-based air-quality monitoring system for everyone, in: Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems, SenSys '14, ACM, New York, NY, USA, 2014, pp. 251–265. doi:10.1145/2668332.2668346.
URL http://doi.acm.org/10.1145/2668332.2668346

[11] H. Jin, L. Su, K. Nahrstedt, CENTURION: incentivizing multi-requester mobile crowd sensing, CoRR abs/1701.01533. arXiv:1701.01533.
URL http://arxiv.org/abs/1701.01533

[12] R. Azzam, R. Mizouni, H. Otrok, A. Ouali, S. Singh, GRS: A group-based recruitment system for mobile crowd sensing, J. Network and Computer Applications 72 (2016) 38–50. doi:10.1016/j.jnca.2016.06.015.
URL https://doi.org/10.1016/j.jnca.2016.06.015

[13] U. ul Hassan, E. Curry, Efficient task assignment for spatial crowdsourcing: A combinatorial fractional optimization approach with semi-bandit learning, Expert Systems with Applications 58 (2016) 36 – 56. doi:https://doi.org/10.1016/j.eswa.2016.03.022.
URL http://www.sciencedirect.com/science/article/pii/S0957417416301014

[14] M. H. Cheung, R. Southwell, F. Hou, J. Huang, Distributed time-sensitive task selection in mobile crowdsensing, in: Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc '15, ACM, New York, NY, USA, 2015, pp. 157–166. doi:10.1145/2746285.2746293.
URL http://doi.acm.org/10.1145/2746285.2746293

[15] A. Capponi, C. Fiandrino, D. Kliazovich, P. Bouvry, S. Giordano, A cost-effective distributed framework for data collection in cloud-based mobile crowd sensing architectures, IEEE Transactions on Sustainable Computing 2 (1) (2017) 3–16. doi:10.1109/TSUSC.2017.2666043.

[16] P. P. Jayaraman, J. Bártolo Gomes, H. Nguyen, Z. S. Abdallah, S. Krishnaswamy, A. Zaslavsky, Scalable energy-efficient distributed data analytics for crowdsensing applications in mobile environments, IEEE Transactions on Computational Social Systems 2 (3) (2015) 109–123. doi:10.1109/TCSS.2016.2519462.

[17] L. Baresi, S. Guinea, D. F. Mendonca, A3Droid: A framework for developing distributed crowdsensing, in: 2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops), 2016, pp. 1–6. doi:10.1109/PERCOMW.2016.7457103.

[18] G. Wood, Ethereum: a secure decentralised generalised transaction ledger, Ethereum Project Yellow Paper 151.

[19] Z. Lu, W. Liu, Q. Wang, G. Qu, Z. Liu, A privacy-preserving trust model based on blockchain for vanets, IEEE Access 6 (2018) 45655–45664. doi:10.1109/ACCESS.2018.2864189.

[20] M. Singh, Service-Oriented Computing [electronic resource]: Semantics, Processes, Agents, ITPro collection, Wiley.
URL https://books.google.ae/books?id=sq52DwAAQBAJ

[21] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, J. Wang, Untangling blockchain: A data processing view of blockchain systems, IEEE Transactions on Knowledge and Data Engineering 30 (7) (2018) 1366–1385. doi:10.1109/TKDE.2017.2781227.

[22] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, A. Hobor, Making smart contracts smarter, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16, ACM, New York, NY, USA, 2016, pp. 254–269. doi:10.1145/2976749.2978309.
URL http://doi.acm.org/10.1145/2976749.2978309

[23] S. Delgado-Segura, C. Tanas, J. Herrera-Joancomartí, Reputation and reward: Two sides of the same bitcoin, Sensors 16 (6). doi:10.3390/s16060776.
URL https://www.mdpi.com/1424-8220/16/6/776

[24] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J. Liu, Y. Xiang, R. H. Deng, Crowdbc: A blockchain-based decentralized framework for crowdsourcing, IEEE Transactions on Parallel and Distributed Systems 30 (6) (2019) 1251–1266. doi:10.1109/TPDS.2018.2881735.

[25] Y. Lu, Q. Tang, G. Wang, Zebralancer: Private and anonymous crowdsourcing system atop open blockchain, in: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), 2018, pp. 853–865. doi:10.1109/ICDCS.2018.00087.

[26] J. Wang, M. Li, Y. He, H. Li, K. Xiao, C. Wang, A blockchain based privacy-preserving incentive mechanism in crowdsensing applications, IEEE Access 6 (2018) 17545–17556. doi:10.1109/ACCESS.2018.2805837.

[27] D. Chatzopoulos, S. Gujar, B. Faltings, P. Hui, Privacy preserving and cost optimal mobile crowdsensing using smart contracts on blockchain, in: 2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS), 2018, pp. 442–450. doi:10.1109/MASS.2018.00068.

[28] S. Wang, A. Taha, J. Wang, Blockchain-assisted crowdsourced energy systems.

[29] M. Yang, T. Zhu, K. Liang, W. Zhou, R. H. Deng, A blockchain-based location privacy-preserving crowdsensing system, Future Generation Computer Systems 94 (2019) 408 – 418. doi:https://doi.org/10.1016/j.future.2018.11.046.
URL http://www.sciencedirect.com/science/article/pii/S0167739X18320909

[30] Solidity (2018).
URL https://solidity.readthedocs.io/en/latest/

[31] Z. Zheng, F. Wu, X. Gao, H. Zhu, S. Tang, G. Chen, A budget feasible incentive mechanism for weighted coverage maximization in mobile crowdsensing, IEEE Transactions on Mobile Computing 16 (9) (2017) 2392–2407. doi:10.1109/TMC.2016.2632721.

[32] M. Z. Marvit, How Crowdworkers Became the Ghosts in the Digital Machine (2019).
URL https://www.thenation.com/article/how-crowdworkers-became-ghosts-digital-machine/

[33] A. Semuels, The internet is enabling a new kind of poorly paid hell (2019).
URL https://www.theatlantic.com/business/archive/2018/01/amazon-mechanical-turk/551192/

[34] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, Cryptography Mailing list at https://metzdowd.com.

[35] A. Biryukov, D. Khovratovich, I. Pustogarov, Deanonymisation of clients in bitcoin p2p network, in: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14, ACM, New York, NY, USA, 2014, pp. 15–29. doi:10.1145/2660267.2660379.
URL http://doi.acm.org/10.1145/2660267.2660379

[36] A. Baliga, Understanding Blockchain Consensus Models, Persistent, 2019.
URL https://pdfs.semanticscholar.org/da8a/37b10bc1521a4d3de925d7ebc44bb606d740.pdf

[37] J. R. Douceur, The Sybil Attack, in: P. Druschel, F. Kaashoek, A. Rowstron (Eds.), Peer-to-Peer Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 251–260.

[38] L. Sweeney, K-anonymity: A model for protecting privacy, Int. J. Uncertain. Fuzziness Knowl.-Based Syst. 10 (5) (2002) 557–570. doi:10.1142/S0218488502001648.
URL http://dx.doi.org/10.1142/S0218488502001648

[39] X. Wang, K. Govindan, P. Mohapatra, Collusion-resilient quality of information evaluation based on information provenance, in: 2011 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, 2011, pp. 395–403. doi:10.1109/SAHCN.2011.5984923.

[40] S. Feit, TCP/IP, McGraw-Hill, Inc., New York, NY, USA, 2000.

[41] Web3j (2018).
URL https://docs.web3j.io/

[42] M. Venanzi, A. Rogers, N. R. Jennings, Crowdsourcing spatial phenomena using trust-based heteroskedastic gaussian processes (2013).
URL https://eprints.soton.ac.uk/354861/

[43] Ethereum gas station (2019).
URL https://ethgasstation.info/

[44] Ethereum price (2018).
URL https://ethereumprice.org/

[45] J. FRANKENFIELD, Proof of work (2019).
URL https://www.investopedia.com/terms/p/proof-work.asp

[46] H. Tang, Y. Shi, P. Dong, Public blockchain evaluation using entropy and topsis, Expert Systems with Applications 117 (2019) 204 – 210. doi:https://doi.org/10.1016/j.eswa.2018.09.048.
URL http://www.sciencedirect.com/science/article/pii/S0957417418306250

[47] J. FRANKENFIELD, Proof of stake (pos) (2019).
URL https://www.investopedia.com/terms/p/proof-stake-pos.asp

[48] S. SETH, Proof of assignment (poa) (2019).
URL https://www.investopedia.com/terms/p/proof-assignment-poa.asp

[49] M. K. Spencer, Ethereum 2.0 and the future of scalable blockchain (2019).
URL https://medium.com/futuresin/ethereum-2-0-and-the-future-of-scalable-blockchain-389cfcc7760c