Design and Development of a Mobile Dapp for Mobile Crowdsensing over EVM-enabled Blockchains

Lorenzo Gigli
lorenzo.gigli@unibo.it
University of Bologna
Italy

Federico Montori
University of Bologna
Italy

Giacomo Galletti
University of Bologna
Italy

Luca Sciullo
University of Bologna
Italy

Luca Bedogni
University of Modena and Reggio Emilia
Italy

ABSTRACT

Mobile crowdsensing (MCS) is a valuable approach for data collection via personal devices, however, it faces challenges in data security and reward to end users. Blockchain is considered a viable addition to MCS, though few real integrations of this kind are deployed. In this paper, we explore the integration of blockchain into MCS, highlighting benefits and architectural adaptation challenges. We then present a novel mobile distributed application (MDapp) that serves crowdsourcers, workers, and verifiers in managing campaigns, data contribution and validation, as well as in the reward process. We also provide a quantitative analysis across different blockchains to highlight the feasibility and economic considerations of this integration.

CCS CONCEPTS

• **Information systems** → **Collaborative and social computing systems and tools**; • **Security and privacy** → *Distributed systems security*; • **Computer systems organization** → *Peer-to-peer architectures.*

KEYWORDS

Mobile Crowdsensing, Blockchain, distributed systems

ACM Reference Format:
Lorenzo Gigli, Federico Montori, Giacomo Galletti, Luca Sciullo, and Luca Bedogni. 2023. Design and Development of a Mobile Dapp for Mobile Crowdsensing over EVM-enabled Blockchains. In *The Fifth ACM International Workshop on Blockchain-enabled Networked Sensor Systems (BlockSys) (BlockSys '23), November 12, 2023, Istanbul, Turkiye.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3628354.3629531

1 INTRODUCTION

Mobile crowdsensing (MCS) is a paradigm that leverages the collective data obtained by personal devices such as smartphones and wearables, and it has emerged as a viable solution for a variety of applications, from urban planning to health monitoring. However, there are some inherent challenges surrounding data reliability, data quality, security, and privacy which limit its full potential and the widespread adoption in more challenging scenarios. In fact, MCS systems usually require geolocalized and timestamped data to contextualize the measurements received. Here different solutions exist, ranging from data obfuscation techniques to anonymization mechanisms. At the same time, users are generally discouraged from providing data without adequate compensation, however, these techniques make the design of a proper reward system for contributing users a nontrivial issue [13].

In recent years, Blockchains have also been studied as an adaptable solution for a plethora of decentralized systems. Blockchain systems work in a Peer-to-Peer (P2P) fashion, in which each node belonging to the network maintains a list of all the transactions performed within the system, where a transaction is a change in the status of the system that is validated through consensus among all nodes in the network. Since its inception with Bitcoin, there have been a plethora of alternatives, such as Ethereum, Solana, IOTA, and many others. With the advent of smart contracts, which allow the deployment of immutable computer programs within a blockchain, the number of possible use cases has exploded. Integrating the Blockchain within an MCS system may bring many interesting advantages, but it comes at the cost of adapting a naturally decentralized architecture to the traditionally centralized MCS architecture. To realize this vision and embody all the advantages of the blockchain, the MCS architecture requires foundational changes which also have to be validated [4]. At the same time, given the possibly heavy load of data that can be written on the blockchain, it is also important to keep transaction costs low.

In this work, we propose a novel architecture that allows the integration of a smart contract-enabled blockchain within an MCS system. Specifically, we first architecturally conceptualize a blockchain-enabled MCS system that encompasses three types of users: (I) the crowdsourcer, which is the stakeholder running the campaign, (II) the worker, which is the user providing the data, and (III) the verifier, which acts as a quality guarantee, by endorsing or discrediting the data provided by the workers. We implement the proposed architecture by developing both the smart contracts and a hybrid Mobile Distributed application (MDapp), which constitutes the frontend for all three user types. The MDapp is integrated with the Metamask wallet and implemented through the Flutter framework. Finally, we analyze the cost of deploying such an application on different commercially available blockchains, to highlight their benefits and drawbacks, and to prove the feasibility of our proposal.

The rest of this paper is structured as follows: Section 2 introduces and discusses related works from literature; Section 3 details the MCS architecture; Section 4 describes how we implemented the MDapp and the MCS system; Section 5 presents the performance evaluation of our proposals and discusses the results; Section 6 concludes this work and considers avenues for future works.

2 RELATED WORKS

Mobile Crowdsensing is actively used in several projects and services, as it enables the collection of data from user-owned devices without the need to build a dedicated infrastructure. Although there are different challenges in MCS, certainly what kind of reward users contributing to the platform receive is key, as users who do not sense an advantage in providing data tend to quit [14]. We divide between monetary and non-monetary rewards, where the former provides a quantifiable compensation for the user's work, while the latter involves other forms of rewards. Monetary rewards can then be further divided into static, considering a flat compensation for the user-provided data, or dynamic, in which the reward may vary due to the freshness of information, area coverage, and alike. Non-monetary rewards can be more diverse, ranging from social incentives [18], intrinsic motives [12], and user self-interest [3]. For a complete discussion on the different available rewarding mechanisms, we refer the reader to [9].

In this work, we focus on monetary rewards, in which the stakeholder has to provide redeemable certificates to the end user, which can be money, a QR code with a coupon, a discount for specific services, and alike. Specifically in this scenario, the main advantages that the blockchain brings are immutability and transparency. The immutability property is needed so that users have the guarantee that neither their rewards nor the owner of the data they provide can be altered. This is why in recent years it has been possible to find a plethora of works that target the blockchain as a viable solution for rewards in MCS. We mention here [7], which presents a blockchain-based (MCS) framework aimed at bolstering privacy and security within MCS applications, such as those found in smart cities, healthcare, and environmental monitoring. This framework incorporates an incentive mechanism that leverages a three-stage Stackelberg game. Other proposals such as [16] leverage the blockchain to better identify users through private keys, in order to build a more trustworthy user base.

Clearly, there are also proposals that leverage the blockchain to offer cryptocurrencies, such as [15]. Here a novel cryptocurrency is built so that it can be granted whenever users report new measurements to the platform. More recent works propose their own blockchain to natively integrate MCS, for instance, the work in [8] proposes BlockSense, which is built on top of Proof-of-Data (PoD), an ad-hoc consensus algorithm that forces miners to check for data quality instead of performing useless computation. The idea of making blockchain miners the actual verifiers of data quality has been adopted by many others [17], but it forces the deployment of a custom blockchain instead of using commercial solutions.

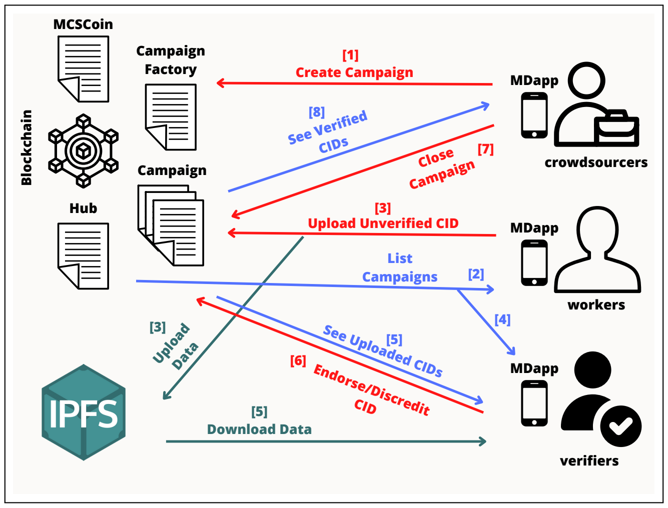


Figure 1: Architecture of the proposed MCS system.

3 ARCHITECTURE

In this section, we present the general architecture of the system with respect to the actors involved. Before delving into the details, it is necessary to introduce the main concepts that power an MCS scenario. First of all, the MCS system involves the exchange of goods between two parties: the **crowdsourcer**, i.e. the entity that requests sensor data, and a crowd of **workers**, each of them possessing the capabilities of collecting such data through their devices. The process of data collection takes place according to a data collection **campaign**, which is published by the crowdsourcer onto an MCS platform and dictates the terms of the data collection process, such as what kind of data is requested, the validity timeframe, and the area of interest. A single contribution by a worker within a campaign is called a **task**. Some MCS scenarios – called pull-based – do not impose a recruitment system, meaning that any worker can contribute to the campaign. This obviously introduces the problem of data quality, which is often tackled by introducing special actors, called **verifiers**, in the loop, especially when the MCS system is implemented on top of a blockchain [1]. Verifiers are then committed to endorsing or discrediting measurements uploaded by workers.

Figure 1 outlines the architecture of our proposed MCS system, which builds upon the interaction among three main architectural systems: a smart contract-enabled blockchain (such as Ethereum), an InterPlanetary File System (IPFS), and a Mobile Dapp, addressed here as MDapp. The blockchain retains a number of smart contracts which act as the backend of our system. Smart contracts can be here seen as immutable computer programs, directly written on-chain and hosting a number of functions. These functions can be executed by any blockchain node through the Ethereum Virtual Machine (EVM), thus we assume that the blockchain is EVM-compatible. This is a fair assumption, as EVM-compatible blockchains are the vast majority [10]. In particular, our system consists of four contracts:

- the **Campaign Contract** corresponds to a single published campaign, thus each campaign is associated with a single Campaign contract. The contract retains metadata about the

campaign that influences whether a data upload is valid or not; then it exposes (i) functions for the worker to upload sensor data, (ii) functions for the verifiers to verify data uploads and (iii) functions for the crowdsourcers to close the campaign and trigger the payoff for all participants.
- the **Campaign Factory Contract** hosts the function for generating a new campaign and, thus, creating the related smart contract. This function is only invoked by crowdsourcers.
- the **Hub Contract** is the entry point to the system, which keeps track of all campaigns and their statuses, along with the function to close an existing campaign and trigger the payoff process for workers and verifiers.
- the **MCScoin Contract** is based on the ERC20 standard and implements the fungible token, called MCSCoin, that makes up for the currency used in the system, along with functions to transfer the ownership of tokens.

Sensor data collected by workers is never saved on-chain, because of its variable size which may trigger significant upload fees (see Section 5 for more details). For this reason, we use IPFS for data storage. IPFS is a P2P distributed file system built upon a network of nodes that retain copies of a subset of the stored files to make them easily sharable [2]. In IPFS every file and directory has a unique address (CID) based on the hash of their content and it is stored into chunks that are organized into a Merkle DAG (Directed Acyclic Graph) object. The CID is then stored on-chain, ensuring that every saved element has a predictable and fixed size.

The MDapp is the user interface through which all three actors in the system interact with the blockchain. In our case, the MDapp is represented by a mobile application that is connected to a wallet in order to send and receive funds, sign and track transactions, and manage authorizations. More in detail, as shown in Figure 1, the interactions among actors and the blockchain take place as follows:

(1) The crowdsourcer creates a campaign by querying the Campaign Factory contract and stakes a definite amount of MCScoins which make up for the cost of the whole campaign.
(2) The worker selects a campaign to participate in by querying the Hub contract.
(3) The worker performs the sensor reading on the end device and uploads the sensed data on IPFS and its CID on the Campaign contract.
(4) The verifier selects a campaign to participate in by querying the Hub contract.
(5) the verifier selects the sensor reading to verify after obtaining the list from the Campaign contract.
(6) the verifier reads the data from IPFS and endorses or disapproves the related CID stored on the Campaign contract.
(7) Once the data collected is sufficient, the crowdsourcer closes the campaign by triggering the related operation on the Hub contract. This operation can be implemented in two ways, as can be seen in Section 5, and results in workers and verifiers sharing the amount of MCScoins staked by the crowdsourcer at the beginning.
(8) the crowdsourcer visualizes all the sensed data from IPFS, once obtained their CIDs from the Campaign contract.

3.1 Costs

The exchange of MCSCoins takes place when a crowdsourcer closes a campaign. More in detail, when a crowdsourcer creates a campaign, she transfers a certain amount of MCSCoins to the campaign address; this amount is known to participants as it is stated publicly in the Campaign contract. According to the default policy (clearly, other policies could be implemented), when the campaign is closed, the amount of MCSCoins staked is subdivided equally among all participating workers, in proportion to the number of endorsed contributions. This may encourage workers to contribute honestly and privilege the campaigns with less data. The same mechanism could be adopted for verifiers (as long as their opinion belongs to the majority) with a second MCSCoins pool, however, this aspect has not been implemented in our proof-of-concept.

Crowdsourcers, workers, and verifiers should also be aware of the inherent costs connected with their operations. All the functions invoked by the actors in Figure 1 have a color coding. The blue calls are view/pure functions, which can be computed by any node in the network and do not require any modification on the chain. Red calls are functions that actually trigger blockchain transactions, and thus are subject to the payment of gas fees on Ethereum and its connected layer 2 blockchains (more details on this aspect are given in Section 5). With this in mind, it is implicit that not only crowdsourcers pay to obtain data but also every data point uploaded by workers as well as every endorse/discredit action performed by a verifier potentially costs gas, which is an additional fee that has nothing to do with our MCSCoin currency.

3.2 Extensions

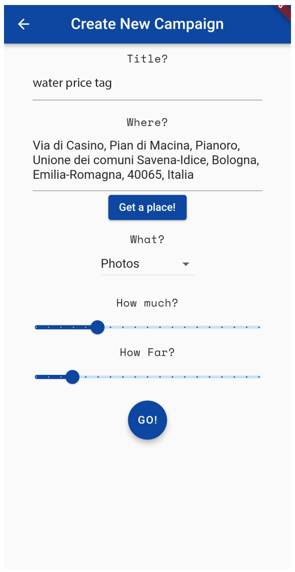
The present paper intends to present a proof-of-concept rather than a production-ready environment, for this reason, we intentionally leave apart some security and privacy aspects that are admittedly very important in the context of MCS but fall out of the scope of this paper. For example, as data is published on IPFS, then anyone can read and steal it, however, there exist solutions for granting cryptography and ensuring data ownership through non-fungible tokens (NFTs) [5]. We also do not deal here with the selection of verifiers such that they cannot perform collusion attacks, as well as with limiting the frequency of contributions by workers. Both these issues can be solved by adding a data oracle layer as in our recent proposal [6].

4 IMPLEMENTATION

We here provide the implementation details of both the smart contracts and the mobile application used in the scenario presented in Section 3. We published open source under GPLv3 license both the code for the smart contracts and the MDapp[1].

The smart contracts mentioned in Figure 1 have all been implemented in Solidity and deployed on the Ethereum testnet. At the time of writing, Ethereum imposes a limit of 24,576 bytes to the size of a single transaction, including the contract itself (the deployment of which is actually a transaction), which is why we decided to separate the system into multiple contracts adopting a factory pattern, where contracts can be deployed automatically through a function of a single factory contract (i.e. Campaign Factory contract).

[1]https://github.com/UniBO-PRISMLab/MDappMobileCrowdSensing



Title?
water price tag

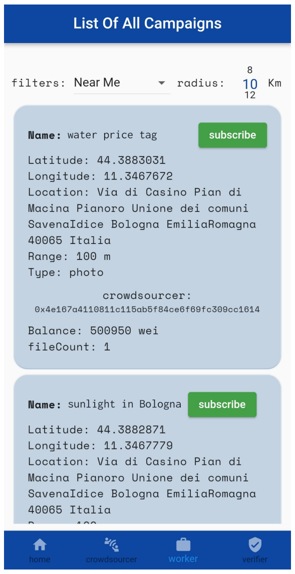
Where?
Via di Casino, Pian di Macina, Pianoro, Unione dei comuni Savena-Idice, Bologna, Emilia-Romagna, 40065, Italia

What?
Photos

How much?

How Far?

(a) Crowdsourcer view: creation of a campaign.

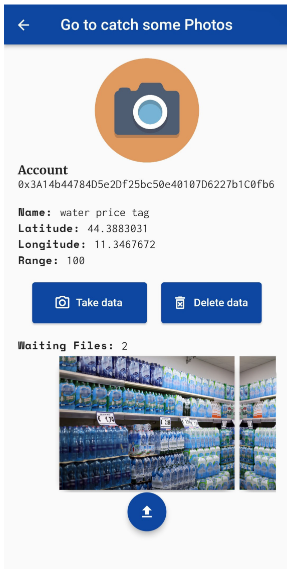


filters: Near Me radius: 8 10 12 Km

Name: water price tag
Latitude: 44.3883031
Longitude: 11.3467672
Location: Via di Casino Pian di Macina Pianoro Unione dei comuni SavenaIdice Bologna EmiliaRomagna 40065 Italia
Range: 100 m
Type: photo
crowdsourcer:
0x4e167a4110811c115ab5f84ce6f69fc309cc1614
Balance: 500950 wei
fileCount: 1

Name: sunlight in Bologna
Latitude: 44.3882871
Longitude: 11.3467779
Location: Via di Casino Pian di Macina Pianoro Unione dei comuni SavenaIdice Bologna EmiliaRomagna 40065 Italia

(b) Worker/Verifier view: list of all campaigns.

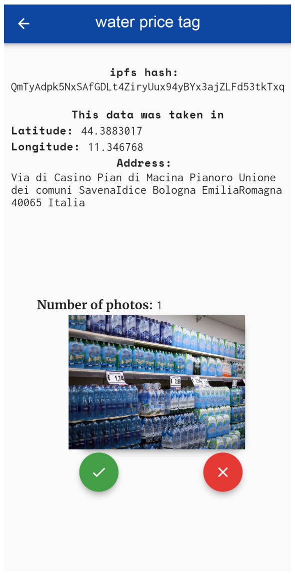


Account
0x3A14b44784D5e2Df25bc50e40107D6227b1C0fb6

Name: water price tag
Latitude: 44.3883031
Longitude: 11.3467672
Range: 100

Waiting Files: 2

(c) Worker view: task execution.



ipfs hash:
QmTyAdpk5NxSAfGDLt4ZiryUux94yBYx3ajZLFd53tkTxq

This data was taken in
Latitude: 44.3883017
Longitude: 11.346768
Address:
Via di Casino Pian di Macina Pianoro Unione dei comuni SavenaIdice Bologna EmiliaRomagna 40065 Italia

Number of photos: 1

(d) Verifier view: display of data to be verified.

Figure 2: The four main screens of the mobile application used by the crowdsourcer, the worker, and the verifier.

4.1 Mobile Distributed Application

We developed a hybrid MDapp using the Dart programming language[2] and the Flutter framework[3]. Flutter allows the creation of applications with the same codebase that can be easily distributed and installed on different operating systems, like Windows, Android, and iOS. For the goal of this paper, we limited the test to Android devices solely. As previously explained, we consider three different types of users for the scenario: (i) the crowdsourcer, (ii) the worker, and (iii) the verifier. Depending on the role, the user can interact with the application in a different way.

As shown in Figure 2a, the crowdsourcer can create a new campaign through a form, inserting the name of the campaign, the interested area with the desired radius, the type of data to be collected in the campaign, and the amount of reward for the entire campaign. The interested area can be selected automatically through the GPS of the mobile device, while for rendering the map and for the manual search of a specific area we rely on the OpenStreetMap[4] APIs. The crowdsourcer can also monitor and manage an active campaign through a dedicated page of the MDapp. More in detail, she can monitor the progress of a campaign, and close the campaign whenever sufficient validated data points have been collected by the workers. The latter operation automatically triggers the reward phase for workers and verifiers. Finally, the crowdsourcer can list the closed campaigns and the data collected for each of them. For the sake of simplicity, and in order to display the collected data, we implemented only a photo carousel in case of image data, and a line chart in case of sensor data. We highlight that further customized visualization methods can be easily added to the applications depending on the type of the collected data. In our example, the crowdsourcer creates a campaign in which workers collect geolocalized photos of price tags of bottled water in supermarkets of different brands with the goal of publishing a survey. This is a popular MCS solution for comparing market retail prices in the absence of online services [11].

A worker has the ability to view all active campaigns, as illustrated in Figure 2b. The worker can filter these campaigns to display only those she has subscribed to or those located in proximity within a specified radius. She can subscribe or unsubscribe to an active campaign and get its details by clicking on the card on the list. A geo-fence service automatically returns the user to the main page if she walks out of the specified area. In this case, the worker gets also informed by the MDapp through a notification that she is not inside the interested area anymore. The MDapp has an additional type of notification for making the worker aware if a campaign to which she subscribed is still open or not. As soon as she subscribes to a new campaign, a background service is instantiated: every 30 seconds it connects to the chain and checks if the campaign was closed. Once the user subscribes to a campaign, the MDapp redirects the user to the dedicated page for uploading the data to collect. Figure 2c shows the worker view for the water price tag campaign. The worker takes the photo and uploads it to IPFS, hence issuing a new transaction on the chain.

The verifier shares the same page for listing the active campaigns with the worker, but after subscribing to a campaign, the MDapp shows the tasks of the campaign, both the ones verified and not verified yet. The verifier can then endorse/discredit a task by clicking on it and accepting or denying the preview of the data collected for such task and downloaded from IPFS, as shown in Figure 2d.

[2]https://dart.dev
[3]https://flutter.dev
[4]https://openstreetmap.org

All the operations that issue a new transaction on the chain – any non-view/pure function call, like the upload of new data, or the verification of a task – must be signed. This process is not directly handled by our MDapp, but instead by a third-party wallet that must be installed and configured beforehand on the same mobile device. More precisely, we used WalletConnect[5], a communication protocol for *web3*, which enables wallets and apps to securely connect and interact. While WalletConnect can work with different wallets, we used Metamask[6], one of the most famous crypto wallets and gateways to the Ethereum blockchain. Finally, in order to interact with the Ethereum blockchain, the MDapp connects to an INFURA[7] endpoint. Infura offers trusted and robust APIs to interact with the Ethereum blockchain, instead of manually performing the operations directly on the blockchain nodes.

5 EVALUATION

In this section, we discuss the evaluation of our proposed system, focusing on the gas consumption of various components of the system and the associated costs across different blockchain platforms. The evaluation aims to provide a clear perspective on the efficiency and feasibility of deploying and operating our mobile crowdsensing system, particularly on popular blockchain platforms such as Ethereum, Polygon, BNB Smart Chain, and Avalanche.

5.1 Experimental Setup

The contracts were developed in Solidity (version 0.8.17)[8] using the Hardhat development environment (version 2.3.3)[9]. To ensure a precise and accurate assessment of gas consumption for contract deployment and function calls, we employed the hardhat-gas-reporter plugin (version 1.0.9)[10]. The plugin was configured to use the CoinMarketCap API[11] to retrieve the cost in USD. For the experiments presented in Section 5.2, the exchange rate was sourced on 19 September 2023. The experiments were conducted on a machine equipped with an Intel Core i7 (6th Gen) 6700HQ processor @ 2.6GHz, with 16GB of RAM, SSD, and Linux Ubuntu 22.04.3 operating system. It is important to note that while we deployed our contracts on the Ethereum Goerli Testnet[12] for running some closed test campaigns, these specific results come from a dedicated test suite built with the Hardhat test framework and executed on the Hardhat Network. Conducting these experiments on a real testnet would have posed challenges due to the need for acquiring test tokens and managing unpredictable network behaviors. The controlled environment of the Hardhat Network provided a consistent and repeatable testing platform, allowing for a more reliable and precise measurement of gas consumption and associated metrics.

Note that the actual payoff has been implemented in two different ways that correspond to two separate paradigms. The first implementation behaves like a real backend, where, by triggering the campaign closure, the crowdsourcer also actively transfers the payment of MCSCoins to all workers (and verifiers) who are meant to be paid (function *closeCampaignAndPay*). With this approach, neither workers nor verifiers will consume resources to claim their reward, however, this operation itself results in high gas costs for the crowdsourcer. The second implementation follows a "Pull" pattern, where the crowdsourcer only triggers the campaign closure (function *closeCampaign*) and divides the initial stake in fair shares. Subsequently, each payee should actively perform a *withdraw* operation to actually obtain the reward. This also divides the gas cost of the rewarding process among the payees instead of charging the entire amount to the payer.

[5]`https://walletconnect.com`
[6]`https://metamask.io`
[7]`https://infura.io`
[8]`https://soliditylang.org`
[9]`https://hardhat.org`
[10]`https://npmjs.com/package/hardhat-gas-reporter`
[11]https://coinmarketcap.com
[12]`https://goerli.net`

5.2 Results

The fundamental metric that dictates the efficiency and feasibility of any Ethereum-based application is its gas consumption. The Campaign Factory contract, essential for crowdsourcers to initiate new campaigns, necessitates 2,896,292 gas units for deployment. In contrast, the MCSCoin contract, which powers the system's economic infrastructure, uses 1,003,567 gas units. Figure 3 shows the associated monetary costs across various blockchain platforms. On Ethereum, deploying the Campaign Factory costs \$115, whereas setting up the MCSCoin costs \$39.85. Polygon offers a more cost-efficient solution with just \$0.27 for the Campaign Factory and a mere \$0.09 for the MCSCoin. The BNB Smart Chain and Avalanche follow with their respective pricing tiers: BNB charges \$1.9 for Campaign Factory and \$0.66 for MCSCoin, while Avalanche requires \$0.88 and \$0.31 for the two. These variations underline the importance of selecting the right platform based on both functional needs and economic considerations.

Similar to deployment costs, the correspondent USD costs associated with invoking the functions across different chains provide a clearer picture of the economic implications of our system. On the Ethereum platform, invoking the *createCampaign* function is the most expensive at \$77.95, followed by *uploadFile* at \$8.42. The other functions, namely *validateFile*, *closeCampaign*, and *withdraw*, cost \$2.58, \$5.15, and \$2.18, respectively. Staying true to its cost-efficient reputation, Polygon demonstrates the lowest costs across all functions, priced below \$0.20. Similarly, the BNB Smart Chain and Avalanche retain their mid-tier cost structure, with all functions costing under \$1.5 and \$0.7, respectively. A visual representation of these costs can be found in Figure 4.

A separate assessment was performed for the *closeCampaignAndPay* function due to its variability based on the number of participants. Gas consumption for this function naturally escalates as the number of participants rises, starting at 202,603 for a single participant and surging to 1,944,129 for 50 participants. Examining the monetary implications on different chains, Ethereum remains the priciest option. For a single user, the cost stands at \$8.06, which increases to \$77.23 for 50 users. The Polygon chain remains the most economical, charging a mere \$0.19 for 50 users. BNB Smart Chain and Avalanche follow suit with their mid-range costs, asking \$1.28 and \$0.60 for 50 users, respectively.

Our evaluation demonstrates clear trade-offs in terms of cost and performance across different blockchain platforms. While Ethereum provides robustness and security, its associated costs might be prohibitive for frequent and large-scale campaigns. On the other

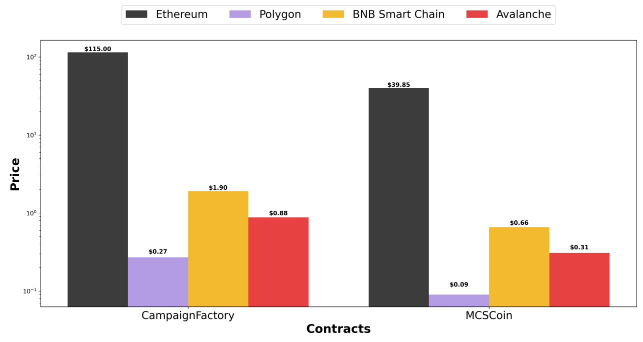


Figure 3: Cost in USD to deploy contracts across chains (logarithmic scale)

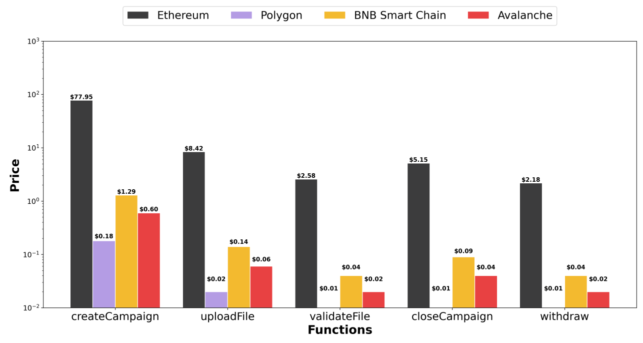


Figure 4: Cost in USD to execute functions across chains (logarithmic scale)

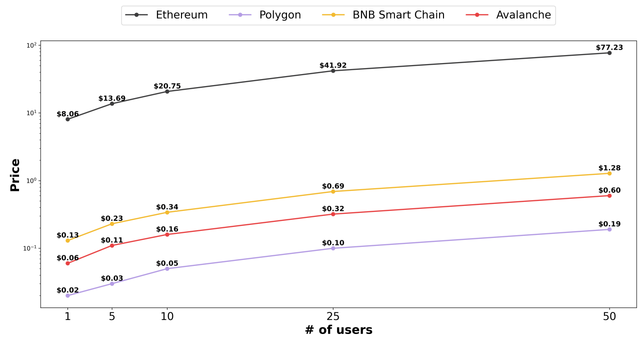


Figure 5: Cost in USD to execute closeCampaignAndPay function across chains with increasing users (logarithmic scale)

hand, chains like Polygon offer almost negligible costs but sacrifice some decentralization and security. Deciding on a platform would require crowdsourcers to carefully consider the scale and frequency of their campaigns. The *closeCampaignAndPay* function provides flexibility, especially for larger entities or public administrations willing to absorb the extra costs for rewarding instead of their users.

6 CONCLUSION

In this paper, we presented a prototype of the integration of an MCS system with blockchain technologies. We first highlighted the architectural implications of such an integration, and then we presented the implementation details of our system, including the smart contracts in the blockchains and the frontend in the form of an MDapp implemented in Flutter. We finally showed the feasibility of our solution by outlining the gas costs in a real deployment. Results clearly show that layer 2 blockchains are able to support our system and scale up. Future works will be dedicated to integrating solutions for privacy, security, and data ownership in this framework.

REFERENCES

[1] Jian An, Jindong Cheng, Xiaolin Gui, Wendong Zhang, Danwei Liang, Ruowei Gui, Lin Jiang, and Dong Liao. 2020. A lightweight blockchain-based model for data quality assessment in crowdsensing. *IEEE Transactions on Computational Social Systems* 7, 1 (2020), 84–97.
[2] Juan Benet. 2014. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561* (2014).
[3] Nirupama Bulusu, Chun Tung Chou, Salil Kanhere, Alex Dong, Shitiz Sehgal, David Sullivan, and Lupco Blazeski. 2008. Participatory sensing in commerce: Using mobile camera phones to track market price dispersion. (01 2008).
[4] Zhiyan Chen, Claudio Fiandrino, and Burak Kantarci. 2021. On blockchain integration into mobile crowdsensing via smart embedded devices: A comprehensive survey. *Journal of Systems Architecture* 115 (2021), 102011.
[5] Marco Di Francesco, Lodovica Marchesi, and Raffaele Porcu. 2023. Kryptosafe: managing and trading data sets using blockchain and IPFS. In *2023 IEEE/ACM 6th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE, 5–8.
[6] Lorenzo Gigli, Ivan Zyrianoff, Federico Montori, Cristiano Aguzzi, Luca Roffia, and Marco Di Felice. 2023. A Decentralized Oracle Architecture for a Blockchain-Based IoT Global Market. *IEEE Communications Magazine* 61, 8 (2023), 86–92.
[7] Jiejun Hu, Kun Yang, Kezhi Wang, and Kai Zhang. 2020. A Blockchain-Based Reward Mechanism for Mobile Crowdsensing. *IEEE Transactions on Computational Social Systems* 7, 1 (feb 2020), 178–191.
[8] Junqin Huang, Linghe Kong, Long Cheng, Hong-Ning Dai, Meikang Qiu, Guihai Chen, Xue Liu, and Gang Huang. 2022. BlockSense: Towards Trustworthy Mobile Crowdsensing via Proof-of-Data Blockchain. *IEEE Transactions on Mobile Computing* (2022).
[9] Luis G. Jaimes, Idalides J. Vergara-Laurens, and Andrew Raij. 2015. A Survey of Incentive Techniques for Mobile Crowd Sensing. *IEEE Internet of Things Journal* 2, 5 (oct 2015), 370–380.
[10] Ruizhe Jia and Steven Yin. 2022. To EVM or not to EVM: Blockchain compatibility and network effects. In *Proceedings of the 2022 ACM CCS Workshop on Decentralized Finance and Security*. 23–29.
[11] Şevval Seray Macakoğlu, Burcu Alakuş Çınar, and Serhat Peker. 2022. Kiyaslio: a gamified mobile crowdsourcing application for tracking price dispersion in the grocery retail market. *International Journal of Web Information Systems* 18, 1 (2022), 55–75.
[12] Emiliano Miluzzo, Nicholas D. Lane, Shane B. Eisenman, and Andrew T. Campbell. [n. d.]. CenceMe – Injecting Sensing Presence into Social Networking Applications. In *Smart Sensing and Context*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–28.
[13] Federico Montori and Luca Bedogni. 2023. Privacy preservation for spatio-temporal data in Mobile Crowdsensing scenarios. *Pervasive and Mobile Computing* 90 (mar 2023), 101755.
[14] Martine Rutten, Ellen Minkman, and Maarten van der Sanden. 2017. How to get and keep citizens involved in mobile crowd sensing for water management? A review of key success factors and motivational aspects. *WIREs Water* 4, 4 (2017).
[15] Jingzhong Wang, Mengru Li, Yunhua He, Hong Li, Ke Xiao, and Chao Wang. 2018. A Blockchain Based Privacy-Preserving Incentive Mechanism in Crowdsensing Applications. *IEEE Access* 6 (2018), 17545–17556.
[16] Weizheng Wang, Yaoqi Yang, Zhimeng Yin, Kapal Dev, Xiaokang Zhou, Xingwang Li, Nawab Muhammad Faseeh Qureshi, and Chunhua Su. 2022. BSIF: Blockchain-Based Secure, Interactive, and Fair Mobile Crowdsensing. *IEEE Journal on Selected Areas in Communications* 40, 12 (2022), 3452–3469.
[17] Jinwen Xi, Shihong Zou, Guoai Xu, and Yueming Lu. 2022. CrowdLBM: A lightweight blockchain-based model for mobile crowdsensing in the Internet of Things. *Pervasive and Mobile Computing* 84 (2022), 101623.
[18] Guang Yang, Shibo He, Zhiguo Shi, and Jiming Chen. 2017. Promoting Cooperation by the Social Incentive Mechanism in Mobile Crowdsensing. *IEEE Communications Magazine* 55, 3 (mar 2017), 86–92.