NF-Crowd: Nearly-free Blockchain-based Crowdsourcing

Chao Li*, Balaji Palanisamy†, Runhua Xu†, Jian Wang* and Jiqiang Liu*
*Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, Beijing, China
†School of Computing and Information, University of Pittsburgh, Pittsburgh, USA

***Abstract*—Advancements in distributed ledger technologies are rapidly driving the rise of decentralized crowdsourcing systems on top of open smart contract platforms like Ethereum. While decentralized blockchain-based crowdsourcing provides numerous benefits compared to centralized solutions, current implementations of decentralized crowdsourcing suffer from fundamental scalability limitations by requiring all participants to pay a small transaction fee every time they interact with the blockchain. This increases the cost of using decentralized crowdsourcing solutions, resulting in a total payment that could be even higher than the price charged by centralized crowdsourcing platforms. This paper proposes a novel suite of protocols called `NF-Crowd` that resolves the scalability issue by reducing the lower bound of the total cost of a decentralized crowdsourcing project to $O(1)$. `NF-Crowd` is a highly reliable solution for scaling decentralized crowdsourcing. We prove that as long as participants of a project powered by `NF-Crowd` are rational, the $O(1)$ lower bound of cost could be reached regardless of the scale of the crowd. We also demonstrate that as long as at least one participant of a project powered by `NF-Crowd` is honest, the project cannot be aborted and the results are guaranteed to be correct. We design `NF-Crowd` protocols for a representative type of project named crowdsourcing contest with open community review (CC-OCR). We implement the protocols over the Ethereum official test network. Our results demonstrate that `NF-Crowd` protocols can reduce the cost of running a CC-OCR project to less than $2 regardless of the scale of the crowd, providing a significant cost benefit in adopting decentralized crowdsourcing solutions.**

I. Introduction

In the recent years, crowdsourcing has been gaining attention as a promising modern business model that enables individuals and organizations to receive services from a large group of people or crowd. Crowdsourcing services support a variety of tasks ranging from software development to logo designs [7]. For example, `LEGO Ideas` is attracting a lot of fan designers to enter prize contests by submitting original proposals for new `LEGO` Ideas sets. A few top-ranked proposals in the contest have resulted in successful commercialization [22]. Likewise, since 2005, UNIQLO has continuously held annual Global T-Shirt Design Competitions (UTGP) and its 14th UTGP in 2019 has attracted over 18,000 entries from all over the world [8]. The increasing popularity of crowdsourcing solutions is also driving the rise of crowdsourcing intermediate platforms such as `Upwork` [33], `99designs` [1] and `designContest` [11] that connect business clients to designers including individual freelancers and design agencies. For instance, via `designContest`, a startup company may set up a logo design contest with a description of its requirements and include a monetary reward. After receiving a large number of entries from interested designers, the client may invite its target audience such as its followers on Twitter to vote for the favorite design and finally pick one or multiple winning entries based on the voting result. Nearly all such crowdsourcing intermediate platforms make profits by charging fees from clients and designers. `Upwork` reported in second quarter 2019 that its Gross Services Volume (GSV) grew 20% year-over-year to $518.8 million [23]. On a $500 crowdsourcing project in `Upwork`, the platform would charge $100 as the service fee. Similarly, `99designs` would charge $75 from a $500 project as the platform fee. Such high service fees significantly increase the cost of running crowdsourcing projects online. Clients and designers have no choice but accept it as the mutually distrusted parties need a trustworthy intermediary for avoiding dishonest behaviors such as free-riding and false-reporting [39].

Recent advancements in blockchain technology have led to the development of numerous open smart contract platforms including Ethereum [4], [36]. Ethereum has become a promising technology for decentralizing traditional centralized online services and as result, in the recent years, we have witnessed a rapid proliferation of numerous decentralized applications including decentralized crowdsourcing systems [5], [14], [18], [26], [27], [34], [38]. Such services offer clients and designers an option to reduce the high intermediary fee required in centralized crowdsourcing systems. However, blockchains provide tamper resistance properties only at a cost. For example, Ethereum charges each transaction a small fee based on the complexity. In Ethereum, tens of thousands of miners follow the Proof-of-Work (PoW) consensus protocol [29] to compete for solving puzzles and each winner receives a monetary reward for packaging the recent transactions (i.e., transferring fund, executing functions or creating smart contracts) into a new block appended to the end of the blockchain. Fees charged from transactions within a new block in the blockchain are paid to the competition winner who packages the block. People trust that no one can tamper with the blockchain as the probability of a single miner to win in several consecutive competitions to be able to change the network consensus about the blockchain state is negligible. As part of the competition reward, transaction fees help incentivize miners to invest more computation resources into the competitions, which in turn increases the difficulty of competitions and improves the overall safety of the blockchain. Also, transaction fees help protect

Ethereum against DDoS and Sybil attacks [13] as the cost of creating n transactions will have a cost of $O(n)$. Despite their significance to the safety of Ethereum, transaction fees turn out to be an obstacle to existing decentralized crowdsourcing systems, especially when crowdsourcing projects are scaled up. Consider that a client sets up a decentralized design contest in Ethereum, where each entry needs to be submitted by a designer via a transaction that costs a small fee, say \$1. The total fee charged in this contest would be cheaper than a contest in `99designs` only when there are less than 75 entries. In other words, intermediary fees are not eliminated by decentralizing crowdsourcing but are paid to a decentralized infrastructure instead of a centralized service provider for the same purpose of acquiring trust. As a result, it is not surprising to see that sometimes there may be no considerable economic advantage in decentralizing crowdsourcing. For example, if we take the price of ether [1] as its mean value during the first half of the year 2019 recorded in *Etherscan* [17], a crowdsourced image tagging task completed via CrowdBC [26] could very well spend over four times the price charged by the centralized Amazon Mechanical Turk [32]. Similarly, aggregating data from 1,000 providers via decentralized crowdsensing [14] could cost up to \$170.

This paper aims at addressing the challenging question: *how to provide a reliable solution that decouples the cost of decentralizing crowdsourcing from the scale of the crowd?* Our research illustrates that the root cause of the high cost in existing decentralized crowdsourcing systems is the lack of cost-efficient solutions to exploit decentralized trust. Crowdsourcing projects usually involve steps that aggregate data (i.e., contest entries or sensed data) from the crowd of scale n or perform calculations on aggregated data (i.e., determine winning entries based on votes). Such steps become cost-intensive in smart contract platforms like Ethereum as they either charge accumulated small transaction fees (TYPE $n \times 1$) or a single large transaction fee (TYPE $1 \times n$), both resulting in $O(n)$ cost. In this paper, we propose a novel suite of protocols called `NF-Crowd` that reliably resolves the scalability issue by reducing the lower bound of the total cost of a decentralized crowdsourcing project to $O(1)$. We prove that as long as participants of a project powered by `NF-Crowd` are rational, the $O(1)$ lower bound of cost could be reached regardless of the scale of the crowd. We also demonstrate that as long as at least one participant of a project powered by `NF-Crowd` is honest, the project cannot be aborted and the results are guaranteed to be correct. We design `NF-Crowd` protocols for a representative type of project named crowdsourcing contest with open community review (CC-OCR). We implement the protocols over the Ethereum official test network. Our results demonstrate that `NF-Crowd` protocols can reduce the cost of running a CC-OCR project to less than \$2 regardless of the scale of the crowd, providing a significant cost benefit in adopting decentralized crowdsourcing solutions.

The rest of this paper is organized as follows: We start by introducing preliminaries in Section II. In Section III, we present a strawman protocol for CC-OCR projects and categorize the cost-intensive steps resulting in $O(n)$ cost. Then, in Section IV, we propose the `NF-Crowd` protocol for CC-OCR projects that reduces the lower bound of cost to $O(1)$. We implement and evaluate the `NF-Crowd` protocols over the Ethereum official test network in Section V. Finally, we discuss related work in Section VI and conclude in Section VII.

[1] The native cryptocurrency in Ethereum, denoted by Ξ.

II. Preliminaries

In this section, we discuss the preliminaries about smart contracts and introduce the key assumptions, key cryptographic tools and notations used in our work. Though we discuss smart contracts in the context of Ethereum [36], our solutions are applicable to a wide range of other smart contract platforms as well.

A. Account types

There are two types of accounts in Ethereum, namely External Owned Accounts (EOAs) and Contract Accounts (CAs). To interact with the Ethereum blockchain, a user needs to create an EOA and control it via a pair of keys. Specifically, the public key can generate a 20-byte address to uniquely identify the EOA and the private key can be used by the user to sign transactions or other types of messages. Then, any user can create a smart contract by sending out a contract creation transaction from a controlled EOA. The 20-byte address of the created smart contract becomes the unique identity of the contract account (CA).

B. Transactions

The state of Ethereum blockchain can only be changed by the external world (i.e., EOAs) using transactions. A transaction is a serialized binary message sent from an EOA (i.e., sender) that contains the following elements:

- *nonce*: a sequence number issued by the EOA (i.e., transaction creator) to prevent transaction replay;
- *gas price*: the price of gas the EOA is willing to pay;
- *gas limit*: the maximum amount of gas the EOA can afford;
- *recipient*: the recipient account address;
- *value*: the amount of ether to send to the recipient;
- *data*: the binary data payload;
- *vrs*: the ECDSA digital signature of the EOA.

Depending on the value at *recipient* (i.e., EOA or CA or 0x0), transactions can be classified into three categories.

Fund transfer transaction: A transaction with an EOA as *recipient* and a non-empty *value* is a fund transfer transaction, which is used to transfer an amount of ether from the sender EOA to the *recipient* EOA. On the other hand, *data* carried by a transaction is usually ignored by Ethereum clients and wallets that help users control their EOAs.

Function invocation transaction: When a transaction involves a CA as *recipient* as well as a non-empty *data*, it is usually a function invocation transaction for calling a

function within an existing smart contract. Specifically, when the transaction is used to call a function with arguments, such as:

```
function f(uint _arg1, uint _arg2) public {}
```

the *data* payload of the transaction would be in the form of

$$encode(`f(uint256, uint256)')|encode(\_arg1)|encode(\_arg2)$$

namely concatenation of the encoded string "$f(uint256, uint256)$", also called as *function selector*, and the encoded values for all function arguments. When a function invocation transaction additionally carries a non-empty *value* and the invoked function is marked *payable* such as:

```
function withdraw(uint _amount) public payable {}
```

The amount of ether indicated by *value* would be transferred from the sender EOA to the *recipient* CA.

Contract creation transaction: In Ethereum, there is a special type of transaction for creating new smart contracts. Such a transaction, usually referred to as a contract creation transaction, carries a special *recipient* address 0x0, an empty *value* and a non-empty *data* payload. A smart contract (or contract) in Ethereum is a piece of program created using a high-level contract-oriented programming language such as *Solidity* [31]. After compiling into a low-level bytecode language called Ethereum Virtual Machine (EVM) code, the created contract is filled into a contract creation transaction as the *data* payload.

A user, after filling *recipient*, *value* and *data* of a transaction, will then input *nonce*, *gas price* and *gas limit* and finally sign the transaction with the private key of the sender EOA to get signature *vrs*. After that, a complete transaction is created. To make the transaction get executed to change the state of the Ethereum blockchain, the transaction should be broadcast to the entire Ethereum network formed by tens of thousands of miner nodes. Following the Proof-of-Work (PoW) consensus protocol [29], miners in Ethereum competitively solve a blockchain puzzle and the winner packages the received transactions into a block and appends the new block to the end of Ethereum blockchain. From then on, it is hard to tamper with the blockchain state updated by the transaction (i.e., transferred fund, executed function or created contract). Thus, transactions and smart contracts in Ethereum are executed transparently in a decentralized manner and the results are deterministic.

C. Transaction fees

In order to either deploy a new contract or call a deployed contract in Ethereum, one needs to spend Gas, or transaction fees. Based on the complexity of the contract or that of the called function, an amount of ether needs to be spent to purchase an amount of Gas as a transaction fee, which is then paid to the winning miner. The Gas system is important for Ethereum as it helps to incentivize miners to stay honest, to nullify denial-of-service attacks and to encourage efficiency in smart contract programming. On the other hand, the Gas system requires protocols, especially the multi-party ones, to be designed with higher scalability in Ethereum. This is due to the fact that even a single-round multi-party protocol could spend a lot of money to run in case of a large number of participants.

TABLE I: Summary of notations.

notation	description
C	a client who sets up a design contest
D	a designer who wants to win a reward
R	a reviewer who casts vote to entries
S	a smart contract
$S.f()$	function $f()$ within contract S
$\rightrightarrows$	broadcast information via off-chain channels
$\dashrightarrow$	transmit infomation via *private* off-chain channels
$\Rightarrow$	invoke a function within a smart contract
addr $(*)$	an address of an EOA or a CA
keccak $(*)$	a keccak hash value
ipfs $(*)$	a content-addressed IPFS link

D. Off-chain channels

In Ethereum, nodes forming the underlying P2P network can send messages to each other via off-chain channels established through the Whisper protocol [35]. By default, messages are broadcast to the entire P2P network. A node can set up a filter to only accept messages marked with a specific 4-byte topic. Besides, a node can locally generate a pair of asymmetric Whisper keys and reveal the public Whisper key to the blockchain, which allows other nodes to privately communicate with it.

E. Key assumptions

We make the following key assumptions in this paper:

- We assume that the underlying blockchain system satisfies *liveness*, *consistency* and *immutability* properties [19].
- We also make the synchrony assumption that ensures that there exists a known upper bound on the delay of messages.
- Finally, we assume that at least one reviewer is honest.

F. Cryptographic tools

The design of `NF-Crowd` protocols employs several key cryptographic tools: (1) we use a standard notion of Merkle trees and denote the function of getting the Merkle root of a tree as $root \leftarrow merkleRoot(elements)$. (2) we use the Keccak 256-bit hash function supported by Ethereum and it is denoted as $keccak(*)$. (3) we use the ECDSA signature supported by Ethereum. Specifically, a EOA (i.e., *signer*) can sign any message via *JavaScript* API and get signature $vrs \leftarrow sig(keccak(message))$. Later, other EOAs or CAs can recover the address of the *signer* EOA (i.e. $addr(signer)$) via *JavaScript* API or *Solidity* native function and get $addr(signer) \leftarrow vf(keccak(message), vrs)$. (4) we use a distributed file system named InterPlanetary File System (IPFS) [3] to cost-efficiently store data to Ethereum blockchain. Specifically, instead of paying a large transaction fee to store a large file on the chain, we could just store the immutable, permanent IPFS link (hash) of the file obtained via $link \leftarrow ipfs\ (file)$ by paying a much smaller fee, which still timestamps and secures the content of file.

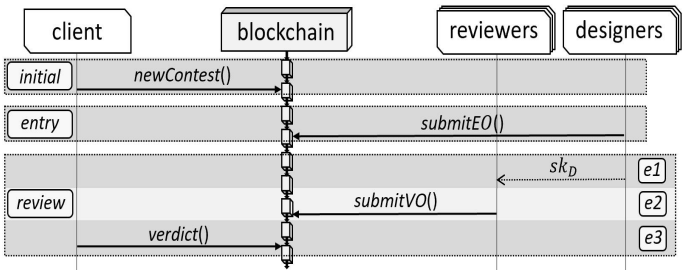

Fig. 1: Strawman CC-OCR protocol sketch. Solid lines denote on-chain transactions. Dotted lines denote off-chain communication.

In the next two sections, we start by presenting a strawman protocol for a CC-OCR crowdsourcing scenario. We then introduce the idea of `NF-Crowd` for reducing cost to $O(1)$. Before discussing the proposed protocols, we summarize the notations that will be used in the rest of this paper in Table I.

III. CC-OCR: A STRAWMAN PROTOCOL

In this section, we first describe the crowdsourcing contest with open community review (CC-OCR) as a three-phase process. We then propose a strawman protocol that decentralizes CC-OCR projects over Ethereum. We finally identify and categorize the cost-intensive steps within the strawman protocol that results in its $O(n)$ total cost.

A. CC-OCR as a three-phase process

We describe a CC-OCR project as a three-phase process similar to the procedure of `LEGO Ideas` [22]:

- *CC-OCR.initial*: The client (C) initiates a design contest with a detailed description of its requirements as well as a reward.
- *CC-OCR.entry*: The interested designers (Ds) within the community submit their entries to the contest.
- *CC-OCR.review*: The client (C) invites selected members from the community as reviewers (Rs) to cast votes to the entries (one vote per R) and pick one or multiple winning entries based on the voting result. Here, the known identities of reviewers prevent them from performing a Sybil attack and hence, the one vote per reviewer could be guaranteed.

B. The strawman CC-OCR protocol

We first propose a strawman protocol to decentralize CC-OCR projects over Ethereum. We sketch the protocol in Fig. 1 and present the formal description in Fig. 2. The regulations of the strawman protocol are programmed as an agency smart contract S_{ag}, through which a client can hold a contest to receive entries and votes from a community.

CC-OCR.initial: Client C initiates a contest by sending out a function invocation transaction from an owned EOA to call the function $newContest()$ at CA $addr(S_{\text{ag}})$. The transaction would carry two arguments at its $data$ payload: (1) a list of $deadlines$ indicating ending times of phases or epochs in the contest; (2) an IPFS link guiding designers to a file describing the requirements of this contest. The transaction would also include a non-empty $value$ to transfer an amount of ether to CA $addr(S_{\text{ag}})$ to reward winning entries. From then on, the arguments are permanently recorded in the blockchain and this new contest can be uniquely identified via $addr(C)$ the client address and cn the number of contests that have been created by C at CA $addr(S_{\text{ag}})$.

CC-OCR.initial:

1. Client creates a contest: $C \Rightarrow S_{\text{ag}}.newContest\ (deadlines, ipfs\ (description), \Xi reward)$.

CC-OCR.entry:

2. Designers Ds submit entry objects (EOs):
   2.1. Each D creates $EO \coloneqq ipfs\ (E(pk_D, proposal))$.
   2.2. $D \Rightarrow S_{\text{ag}}.\ submitEO([addr(C), cn], EO)$.

CC-OCR.review:

3. Each $D \rightrightarrows: sk_D$.
4. Reviewers Rs submit vote objects (VOs):
   4.1. Each R creates $VO \coloneqq addr(D)$.
   4.2. $R \Rightarrow S_{\text{ag}}.\ submitVO([addr(C), cn], VO)$.
5. Client reveals final verdict: $C \Rightarrow S_{\text{ag}}.\ verdict(cn)$.

Fig. 2: Strawman CC-OCR protocol

CC-OCR.entry: After reading the descriptions, interested designers from the community could submit entries to the contest. The protocol requires a designer D to create an entry object *EO* by first generating a one-time asymmetric key pair, then encrypting the detailed $proposal$ with the public key pk_D and finally computing the IPFS link of encrypted $proposal$. Such an *EO* could make $proposal$ confidential before the review phase and also reduce the size of data stored on the chain. After that, the designer could call $submitEO()$ with a transaction carrying arguments *EO* as well as $[addr(C), cn]$ that specifies the participating contest. It is worth noting that any entry submmitted after the $deadline$ specified by C for the entry phase would be rejected because $submitEntry()$ includes a time checker function:

```
require(now < deadline);
```

which requires miners to only accept the transaction if the timestamp is smaller than $deadline$ input via $newContest()$.

CC-OCR.review: The protocol divides the review phase into three epochs. The first epoch is for designers to reveal private keys sk_D so that reviewers can read their entries. Then, during the second epoch, each reviewer R creates a vote object *VO* which in the strawman protocol is simply the address of the designer that R wants to vote for. The created *VO* would be submitted to CA $addr(S_{\text{ag}})$ via $submitVO()$. Finally, client C shall call $verdict()$ during the third epoch and the function would traverse the votes received by all the entries and pick one or multiple entries obtaining the greatest number of votes as winners, who could later withdraw the $\Xi reward$ deposited by C via $newContest()$.

C. The cost-intensive steps

Despite the simplicity, the strawman CC-OCR protocol involves two key representative types of cost-intensive steps, defined as TYPE $n \times 1$ and TYPE $1 \times n$, respectively.

Definition 1 (TYPE $n \times 1$)**.** *A protocol step is said to be in TYPE $n \times 1$ if the number of transactions created at this step increases along with the scale of the crowd.*

Definition 2 (TYPE $1 \times n$)**.** *A protocol step is said to be in TYPE $1 \times n$ if the cost of a single transaction created at this step increases along with the scale of the crowd.*

It is easy to see that both step 2 and 4 of the strawman protocol in Fig. 2 are in TYPE $n \times 1$ while step 5 is in TYPE $1 \times n$. At step 2 (4), data is aggregated from the crowd and each designer (reviewer) needs to submit data via a function invocation transaction $submitEO()$ ($submitVO()$) that spends a certain amount of ether and hence, the accumulated cost at this step increases along with the scale of the crowd (i.e., number of transactions). At step 5, function $verdict()$ contains a *for* loop to iterate over all entries to pick the winners and each iteration spends a certain amount of ether and therefore, the accumulated cost at this step also increases along with the scale of the crowd (i.e., number of iterations).

Theorem 1. *A TYPE $1 \times n$ step is associated with at least one TYPE $n \times 1$ step, but not vice versa.*

We can see that even though steps 2 and 4 are in TYPE $n \times 1$, there is a difference between them. Specifically, step 2 leverages the blockchain as a bulletin board to post entries that meet the requirements of the contest (e.g., before $deadline$) in a deterministic and trustworthy way and hence, data received at step 2 (i.e., *EO*) is not associated with any future step in the protocol. In contrast, data aggregated at step 4 (i.e., *VO*) is associated with step 5 as the votes are collected for the purpose of being the inputs of a function that outputs poll winners.

Next, we introduce the NF-crowd CC-OCR protocol that leverages cost-cutting strategies to convert the two types of cost-intensive steps into cost-efficient steps and reduce the lower bound of the total cost of a decentralized CC-OCR project to $O(1)$.

IV. CC-OCR: An NF-Crowd protocol

We begin by presenting the high-level ideas of the NF-Crowd CC-OCR protocol. We then present the strategies of cutting costs at TYPE $n \times 1$ steps and TYPE $1 \times n$ steps in detail. We finally analyze the total cost and safety of the proposed protocol. We present the NF-Crowd CC-OCR protocol sketch in Fig. 3 and the formal description in Fig. 4.

A. High-level ideas

The NF-Crowd CC-OCR protocol design includes the following novel mechanisms:

Enforceable off-chain execution: Similar to the strawman CC-OCR protocol, most existing decentralized crowdsourcing systems are designed to be executed in an on-chain mode, where both data storage and computation over stored data are performed on the blockchain to employ the decentralized trust in a simple but expensive way [14], [26], [37]. We believe that decentralized trust could be employed in a more cost-effective way. Specifically, we design the NF-Crowd CC-OCR protocol to be executed in an off-chain mode by default, involving no cost-intensive on-chain operations resulting in $O(n)$ cost when all participants are honest. In case if any dishonest participant performs any fraudulent behavior that violates the protocol and aborts the off-chain execution, any honest participant reserves the ability to switch the off-chain mode to an on-chain mode similar to the strawman CC-OCR protocol so that the execution of the protocol could always get enforced.

Punishable protocol violation: In order to incentivize participants to stay honest so that the protocol can end successfully in its off-chain mode, we require each participant to lock an amount of ether in smart contracts as a security deposit to penalize potential misbehaviors that violate the protocol by confiscating the security deposit paid by the corresponding dishonest participant. Specifically, anyone who wishes to join the community to engage in a contest needs to first send out a function invocation transaction to call a function called $joinCommunity()$ at CA $addr(S_{\text{ag}})$. The transaction carries no arguments but is with a non-empty $value$ to transfer an amount of ether to the CA as $\Xi deposit$. Likewise, during the initial phase, besides $\Xi reward$, the $newContest()$ function also charges an amount of ether from client C as $\Xi deposit$.

B. TYPE $n \times 1$ cost-cutting strategy

The proposed strategy for cutting the cost of TYPE $n \times 1$ steps consists of four components, illustrated as step 2.1 (4.1), step 2.2 (4.2), step 2.3 (4.3) and step 6 in Fig. 4, respectively.

Off-chain aggregation: The first component converts TYPE $n \times 1$ on-chain aggregation into TYPE $1 \times n$ off-chain aggregation. The on-chain aggregation employed in the strawman protocol requires participants (i.e., designers or reviewers) to separately submit data to the blockchain, resulting in n transactions. In the NF-Crowd protocol, participants transmit data to a single uploader (i.e., client C), who then submits aggregated data via a single transaction to the blockchain. In Fig. 4, at step 2.1 (4.1), each *EO* (*VO*) additionally carries a signature vrs_D (vrs_R) and is then transmitted to client C via off-chain channels.

On-chain Merkle root: The second component leverages a Merkle tree to reduce the cost of TYPE $1 \times n$ off-chain aggregation from $O(n)$ to $O(1)$. Instead of uploading raw aggregated data to the blockchain, client C here can group the aggregated data as a number of chunks[2], create a Merkle tree that takes the chunks as elements, make chunks public via off-chain channels and upload the Merkle root to the blockchain. This strategy has the following properties:

- *O(1) cost*: The only data that needs to be uploaded is the Merkle root and the cost of uploading the 32-byte root is constant.

[2] Aggregated data is grouped into chunks for reducing the cost of reloading aggregated data onto the chain. We discuss more details on this in Section V.

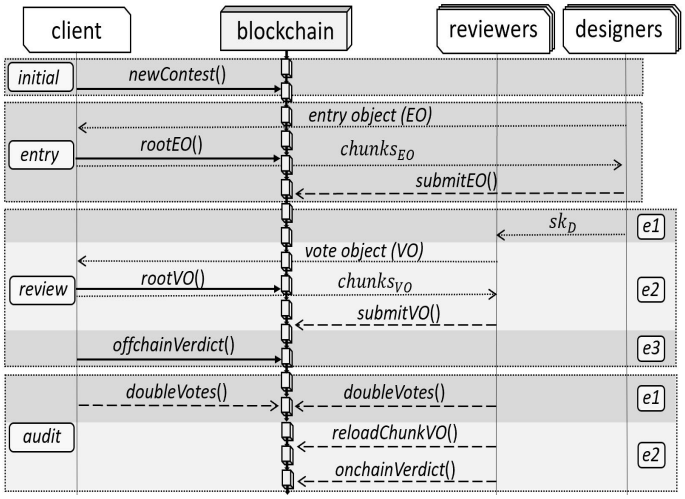

Fig. 3: NF-Crowd CC-OCR protocol sketch. Solid lines denote on-chain transactions. Dotted lines denote off-chain communication. Dashed lines denote available on-chain transactions as countermeasures.

- *Transparency*: All chunks are available to all participants via off-chain channels.
- *Tamper resistance*: The integrity of each chunk could be verified via the on-chain Merkle root and the integrity of each *EO (VO)* could be verified via vrs_D (vrs_R).
- *Traceability*: Signatures vrs_D (vrs_R) could be used to obtain the addresses of the designers (reviewers).

Thus, as long as all the participants are honest, this proposed strategy can achieve $O(1)$ cost with the same level of security as in the on-chain aggregation strategy. At step 2.2 (4.2) in Fig. 4, client C creates a Merkle tree for received *EOs* (*VOs*), upload Merkle root via function *rootEO*() (*rootVO*()) and reveal the IPFS link of $chunks_{EO}$ ($chunks_{VO}$) via off-chain channels.

Countermeasure against intentional exclusion: In on-chain aggregation, data is uploaded by participants themselves and it is hard to keep any participant from getting involved in a contest. In off-chain aggregation, for the purpose of cutting costs, data from all participants is uploaded together by a single uploader, hence a dishonest uploader here has the ability to intentionally exclude data belonging to certain participants from the Merkle tree. As a countermeasure strategy, when a designer (reviewer) fails to verify her data via the on-chain Merkle root, the designer (reviewer) could re-upload the data onto the chain by herself as in the on-chain aggregation before the end of entry (review.e2) phase. The additional cost is shared between the participant and the uploader. In other words, we design the off-chain aggregation strategy to be backed up by the on-chain aggregation strategy and hence a dishonest uploader can hardly block any participant. Meanwhile, both the uploader and participants are incentivized to honestly follow the off-chain strategy to avoid additional charges in the on-chain strategy. This component is illustrated

CC-OCR.initial:

1. Client creates a contest: $C \Rightarrow S_{\text{ag}}.newContest$ $(deadlines,$ *ipfs* $(description),$ [$\Xi deposit$, $\Xi reward$]).

CC-OCR.entry:

2. Designers Ds submit entry objects (EOs):
   - 2.1. D creates $EO \coloneqq [vrs_D,$ *ipfs* $(E(pk_D, proposal))]$, where $vrs_D \leftarrow sig(keccak($*ipfs* $(E(pk_D, proposal))))$. Then, $D \dashrightarrow C$: $[EO]$.
   - 2.2. C organizes received EOs as a Merkle tree and computes $root_{EO} \leftarrow merkleRoot(chunks_{EO})$:
     - 2.2.1. $C \Rightarrow S_{\text{ag}}.\ rootEO(cn, root_{EO})$.
     - 2.2.2. $C \rightrightarrows$: *ipfs* $(chunks_{EO})$.
   - 2.3. $< Countermeasure\ against\ intentional\ exclusion >$ Designer $D \Rightarrow S_{\text{ag}}.submitEO([addr(C), cn], EO)$. The transaction fee shall be shared by C and D.

CC-OCR.review:

3. Each $D \rightrightarrows: sk_D$.
4. Reviewers Rs submit vote objects (*VOs*):
   - 4.1. R creates $VO \coloneqq [vrs_R, addr(D)]$, where $vrs_R \leftarrow sig(keccak(addr(D)))$. Then, $R \dashrightarrow C$: [*VO*].
   - 4.2. C organizes received *VOs* as a Merkle tree and computes $root_{VO} \leftarrow merkleRoot(chunks_{VO})$:
     - 4.2.1. $C \Rightarrow S_{\text{ag}}.$ *rootVO*$(cn, root_{VO})$.
     - 4.2.2. $C \rightrightarrows$: *ipfs* $(chunks_{VO})$.
   - 4.3. $< Countermeasure\ against\ intentional\ exclusion >$ Reviewer $R \Rightarrow S_{\text{ag}}.$*submitVO*$([addr(C), cn],$ *VO*$)$. The transaction fee shall be shared by C and R.
5. $C \Rightarrow S_{\text{ag}}.$*offChainVerdict*$(cn, winners)$.

CC-OCR.audit:

6. $< Countermeasure\ against\ double\ votes >$ C (or R) $\Rightarrow S_{\text{ag}}.$*doubleVotes*$(addr(C), cn,$ *proof*$, chunk_{VO}, i)$.
7. $< Countermeasure\ against\ incorrect$ *off-chain* $computation >$ $R \Rightarrow S_{\text{ag}}.$*reloadChunkVO*$(addr(C), cn,$ *proof*$, chunk_{VO})$. $R \Rightarrow S_{\text{ag}}.$*onchainVerdict*$(addr(C), cn)$.

Fig. 4: NF-Crowd CC-OCR protocol

at step 2.3 (4.3) in Fig. 4, where function $submitEO()$ (*submitVO*()) would refund half of the transaction fee to the designer (reviewer) from $\Xi deposit$ paid by client C.

Countermeasure against double votes: It is possible that a dishonest participant uploads data twice, first time via off-chain aggregation and second time via on-chain aggregation. For instance, a dishonest reviewer can first submit a *VO* to client at step 4.1 and later submit the same *VO* at step 4.3, even if the *VO* could be correctly verified through the on-chain Merkle root. Without taking care of this, the vote may be counted twice, which violates the 'per vote per reviewer' rule. It is hard for contract S_{ag} to detect the double-vote misbehavior because the contract has no knowledge of the off-chain chunks. Even if S_{ag} knows all chunks, it would be quite expensive to verify the double-vote misbehavior on the chain. Therefore, we

Algorithm 1: The doubleVotes() function

```
Input : C, cn, proof, chunk, i.
1  verifyTimestamp(now is in audit.e1);
2  hash ← keccak256(chunk);
3  root ← retrieveRoot(C, cn);
4  if verifyMerkleProof(proof, root, hash) == TRUE then
5  |   (v, r, s, D) ← splitChunk(chunk, i);
6  |   R ← vf(keccak256(D), v, r, s);
7  end
```

Algorithm 2: The reloadChunkVO() function

```
Input : C, cn, proof, chunk.
1  verifyTimestamp(now is in audit.e2);
2  hash ← keccak256(chunk);
3  root ← retrieveRoot(C, cn);
4  if verifyMerkleProof(proof, root, hash) == TRUE then
5  |   for i = 0; i < size(chunk); i++ do
6  |   |   (v, r, s, D) ← splitChunk(chunk, i);
7  |   |   R ← vf(keccak256(D), v, r, s);
8  |   |   reload(R, D);
9  |   end
10 end
```

design a *doubleVotes*() function that could be called by either client C or any reviewer R to detect such a misbehavior off the chain and report it with a *proof* to S_{ag} during epoch-1 of the audit phase that follows the review phase so that the misbehavior could be efficiently verified by S_{ag} on the chain. We show the pseudo-code of *doubleVotes*() at Algorithm 1. The function first verifies the timestamp of the transaction is within epoch-1 of the audit phase (line 1). Then, it computes the hash of the input chunk (i.e., *chunk*) that contains the repeated *VO*, retrieves the Merkle root uploaded by client C at step 4.2 and verifies the input Merkle proof (i.e., *proof*) (line 2-4). After that, function $splitChunk()$ would retrieve the repeated *VO* from the chunk (i.e., i^{th} *VO* in the chunk) and decompose that *VO* to vrs_R and $addr(D)$ (line 5), from which the contract gets the address of R who voted D via off-chain aggregation (line 6). Finally, if the contract finds that R has also voted at step 4.3, it will mark R as dishonest and record the address of the reporter and remove the votes cast by R from the poll. Later, the reporter could withdraw an award confiscated from $\Xi deposit$ paid by the violator.

C. TYPE $1 \times n$ cost-cutting strategy

Our strategy of cutting the cost of TYPE $1 \times n$ steps consists of two components, illustrated as step 5 and step 7 in Fig. 4, respectively.

Off-chain computation: By default, the NF-Crowd protocol encourages computations to be performed off the chain. Therefore at step 5 in Fig. 4, client C could compute the winners off the chain after combining *VOs* received at step 4.2 and *VOs* uploaded at step 4.3 (if any) and simply upload the addresses of winning designers onto the chain via function *offChainVerdict()*. If no one challenges the results in a certain period of time, the contract would take the results as the final verdict. In this way, the expensive on-chain computation could be offloaded to the off-chain side and the cost of uploading computation results is usually a small constant value.

Countermeasure against incorrect off-chain computation: It is important that the offloaded computation could be reloaded onto the blockchain so that off-chain computation results could be replaced with trustworthy on-chain computation results at any moment. By always backing up off-chain computation with on-chain computation, incorrect off-chain computation results would never be adopted in the final verdict. The procedure of reloading off-chain computation consists of two steps. First, as presented in Theorem 1, a TYPE $1 \times n$ step is associated with at least one TYPE $n \times 1$ step, so that any data offloaded via the TYPE $n \times 1$ cost-cutting strategy presented in the Section 4.2 needs to be reloaded onto the chain. After that, computations could be performed on the reloaded data just as in the strawman protocol. For instance, at step 7 in Fig. 4, a reviewer R decides to challenge $winners$ uploaded by client C at step 5. To do this, the reporter should first reload all $chunks_{VO}$ onto the chain via function *reloadChunkVO*(). We show the pseudo-code of *reloadChunkVO*() at Algorithm 2. The function first verifies the timestamp (line 1) and Merkle proof (line 2-4). Then, the function runs a *for* loop to traverse *VOs* included in the input chunk (line 5-9), recover the address of reviewer signing each *VO* (line 6-7) and reload all the votes inside the chunk onto the chain (line 8). Then, the computation could be re-done on the chain via function *onchainVerdict*(). If the results of *onchainVerdict*() are different from the ones in *offchainVerdict()*, the final verdict would take the results of *onchainVerdict*() and a part of $\Xi deposit$ paid by client C would be transferred to the reporter as an award. Otherwise, the final verdict would still take the results of *offchainVerdict()*.

D. Cost and Security analysis

We analyze the cost and security of the proposed NF-Crowd CC-OCR protocol as follows:

Lemma 1. The NF-Crowd CC-OCR protocol has a total cost in the range of $[O(1), O(n)]$ and the $O(1)$ lower bound could be reached as long as the participants are rational.

Proof. The total cost would reach the upper bound c_{ocr}^{up} when (1) all *EOs* are uploaded via $submitEO()$ onto the blockchain and (2) all *VOs* are uploaded via *submitVO*() or *reloadChunkVO*() onto the blockchain and (3) function *doubleVotes*() is called for each *VO* and (4) function *onchainVerdict()* is invoked, namely,

$$O(c_{ocr}^{up}) \rightarrow O(c_{eo} \cdot p_d \cdot n + (c_{vo} + c_{dv}) \cdot p_r \cdot n + c_{ov} + c_{rest}) \rightarrow O(n),$$

where c_{eo} and c_{vo} denotes cost of uploading per *EO* and *VO*, p_d and p_r denotes percentage of designers and reviewers in the crowd that engage in the contest, c_{dv} and c_{ov} express cost of *doubleVotes*() and *onchainVerdict()* and finally c_{rest} represents the total cost of calling other functions inside S_{ag}. In contrast, the total cost would reach the lower bound c_{ocr}^{low} when none of *submitEO*(),*submitVO*(), *reloadChunkVO*(), *doubleVotes*() and *onchainVerdict()* have been invoked, namely $O(c_{ocr}^{low}) \rightarrow O(c_{rest}) \rightarrow O(1)$. All the five functions are

countermeasures against dishonest participants by fixing problems made by them and confiscating $\Xi deposit$ paid by them, so the violators would gain no positive benefit but only a negative payoff. Considering that rational adversaries choose to violate protocols only when doing so brings them a positive payoff [12], [20], [21], [30], rational participants would not choose to lose $\Xi deposit$, which in turn would push the total cost to reach its lower bound. □

We define the abortion of protocols to be the case that neither *offchainVerdict()* nor *onchainVerdict()* has been executed by the end of audit phase. We define the correction of results to be the case that the effect of all the three identified protocol violations (*intentional exclusion*, *double vote* and *incorrect off-chain computation*) to the results has been eliminated by the end of audit phase.

Lemma 2. The NF-Crowd CC-OCR protocol cannot be aborted and the results are guaranteed to be correct as long as at least one reviewer is honest.

Proof. We have assumed that at least one reviewer is honest in Seection II-E. Therefore, in the worst case, the client and all but one reviewers are dishonest. We first prove the never-abort property. During the review phase, the client may refuse to call *offchainVerdict()*, resulting in *incorrect off-chain computation*. As a countermeasure, the honest reviewer could always reload all votes via *reloadChunkVO()* and pick the winners via *onchainVerdict()*. Therefore, with the existence of a single honest reviewer, it is guaranteed that either *offChainVerdict()* or *onchainVerdict()* will be called by the end of audit phase. Next, we prove the always-correct property. As long as there is a single honest reviewer, this reviewer always reserves the ability to include her vote into the pool via *submitVO()* (*intentional exclusion*), to eliminate all illegal on-chain votes via *doubleVotes()* (*double vote*), to reload all legal off-chain votes onto the chain via *reloadChunkVO()* and finally to pick the winners from the pool formed by all legal votes via *onchainVerdict()* (*incorrect off-chain computation*). Therefore, the effect of all the three identified protocol violations to the results can be eliminated before end of audit phase. □

V. Implementation and evaluation

In this section, we present the evaluation of the two protocols: strawman CC-OCR and NF-Crowd CC-OCR. We programmed a smart contract S_{agency} for each of the protocols using *Solidity* and evaluated the protocols over the Ethereum official test network *kovan* [25]. Similar to recent work on blockchain-based platforms [10], [16], the key focus of our evaluation is on measuring gas consumption as the execution complexity and monetary cost in Ethereum are measured via gas consumption. In addition, we compare the cost of the proposed protocols with that of two existing solutions.

A. Gas consumption of proposed protocols

In Table II, we list the key functions in the programmed contracts that interact with protocol participants during different phases and the cost of these functions in both Gas and USD.

TABLE II: Key functions and their cost in Gas and USD. A function marked $*$, $\bullet$ or $\overset{*}{\bullet}$ represents the function is included in strawman CC-OCR, NF-Crowd CC-OCR or both the two protocols, respectively.

Phase	Function	Description	Gas	USD
initial	$\overset{*}{\bullet}$*newContest*	creat a contest	182909/ 244434	\$0.53/ \$0.71
entry	$\bullet$*rootEO*	submit EO root	45322	\$0.13
	$\overset{*}{\bullet}$*submitEO*	submit one EO	143978	\$0.42
review	$\bullet$*rootVO*	submit VO root	65956	\$0.19
	$\overset{*}{\bullet}$*submitVO*	submit one VO	62267	\$0.18
	$\bullet$*offChainVerdict*	submit results of off-chain verdict	44967	\$0.13
	$*$*verdict*	directly execute on-chain verdict	37227+ $2171n_d$	\$0.11+ $0.006n_d$
audit	$\bullet$*doubleVotes*	report double vote	65844	\$0.19
	$\bullet$*reloadChunkVO*	reload VO chunks to blockchain	37843+ $36578n_{vo}$	\$0.11+ $0.11n_{vo}$
	$\bullet$*onchainVerdict*	redo verdict with smart contract	46324+ $2171n_d$	\$0.14+ $0.006n_d$

For ease of expression, a function marked $*$, $\bullet$ or $\overset{*}{\bullet}$ represents the function is included in strawman CC-OCR, NF-Crowd CC-OCR or both the two protocols, respectively. The cost in USD is computed through $cost(USD)$=$cost(Gas) * GasToEther * EtherToUSD$, where *GasToEther* and *EtherToUSD* are taken as their mean value during the first half of the year 2019 recorded in *Etherscan* [17], which are $1.67 * 10^{-8}$ Ether/Gas and 175 USD/Ether, respectively. We next evaluate the cost of the two protocols:

Strawman CC-OCR protocol: The cost in Strawman CC-OCR includes the following components: (1) a client to set up a new contest via $newContest()$ (\$0.53) during *CC-OCR.initial*; (2) each interested designer to submit an entry via $submitEO()$ (\$0.42) during *CC-OCR.entry*; (3) each reviewer to cast a vote via *submitVO*() (\$0.18) during *CC-OCR.review* and (4) client to pick winners via $verdict()$. It thus costs about $\$(0.53 + 0.42n_d + 0.18n_r + (0.11 + 0.006n_d)) = \$(0.64 + 0.426n_d + 0.18n_r)$ for completing a contest that involves n_d designers and n_r reviewers, where $\$(0.11+0.006n_d)$ is the average cost of picking winners among n_d designers in $verdict()$.

NF-Crowd CC-OCR protocol: The lower bound of the cost in NF-Crowd CC-OCR also consists of four parts: (1) a client to set up a new contest via $newContest()$ (\$0.71) during *CC-OCR.initial*, which is more expensive because of the additionally transferred $\Xi deposit$; (2) the client to upload the Merkle root of EOs via $rootEO()$ (\$0.13) during *CC-OCR.entry*; (3) the client to upload the Merkle root of VOs via $rootVO()$ (\$0.19) during *CC-OCR.review* and finally (4) the client to upload winners via *offChainVerdict ()* (\$0.13). The lower bound of the cost is then $\$(0.71 + 0.13 + 0.19 + 0.13) = \$1.16$. If any misbehavior occurs, the countermeasure functions (i.e., $submitEO()$, *submitVO*(), $doubleVotes()$, *reloadChunkVO*(), *onchainVerdict*()) can be invoked and the cost for calling them will be mainly deducted from $\Xi deposit$ paid by protocol violators. It is worth noting that the cost

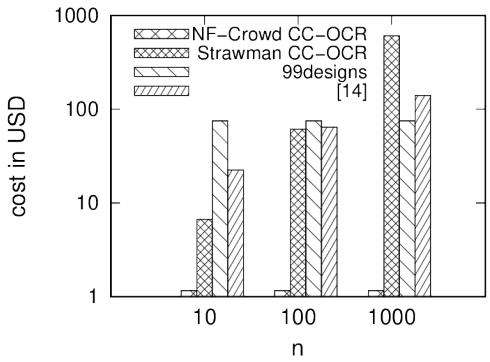

Fig. 5: Cost of the proposed protocols and existing solutions

of function *reloadChunkVO*(), $\$(0.11 + 0.11n_{vo})$, increases along with the number of VOs carried by a $chunk_{VO}$ (i.e., n_{vo}). Grouping *VOs* into $chunks_{VO}$ could effectively reduce the cost of reloading *VOs*. For instance, when $n_{vo} = 1$, the cost of reloading 100 *VOs* would be $\$100 * (0.11 + 0.11 * 1) = \$22$, which is almost twice of $\$(0.11 + 0.11 * 100) = \$11.11$ with $n_{vo} = 100$.

B. Comparison between NF-Crowd and existing solutions

Next, we compare the cost of strawman CC-OCR and NF-Crowd CC-OCR with both the platform fee charged by centralized `99designs` platform and the cost of a recent decentralized protocol in [14]. We assume that the design contest has a \$500 reward. Also, for the purpose of evaluating the scalability of the solutions, we changed the scale of the crowd n and displayed the results in Fig. 5. As can be seen, when n is increased from 10 to 1000, the cost of strawman CC-OCR linearly increases from \$6.7 to over \$600. In contrast, the cost of NF-Crowd CC-OCR stays at \$1.16 constantly regardless of the scale of the crowd. The cost of `99designs` stays at \$75, namely 15% of \$500 reward. In fact, from \$8 reward onwards it is more economical to use NF-Crowd CC-OCR than `99designs`. The cost of the decentralized protocol in [14] increases from \$22.4 to \$140.5 when the crowd gets scaled up. Therefore, we see that NF-Crowd protocols minimize the cost by decoupling the amount of service fees from both the scale of the crowd and the amount of reward.

VI. Related work

A. Centralized crowdsourcing

Crowdsourcing has been emerging as a successful business model and has driven the rise of centralized crowdsourcing platforms such as `Upwork` [33], Amazon Mechanical Turk [32], `99designs` [1] and `designContest` [11]. Via these platforms, clients publish human intelligence tasks (HIT) that are challenging for computers but easy for human to complete with rewards and interested workers (or freelancers or designers) accomplish the tasks to earn rewards. As the primary revenue stream, service fees are charged from the rewards by most of these platforms. `Upwork` [33] charges workers a sliding service fee based on the lifetime billings with a specific client, which consists of 20% of the first \$500, 10% of the billings between \$500 and \$10000 and 5% of the billings that exceed \$10000. For instance, Amazon Mechanical Turk (AMT) [32] charges clients 20% fee on the reward and bonus amount (if any) clients pay workers. Currently, Clients and designers are used to accepting the high service fees as they need a relatively trustworthy intermediary to exclude dishonest behaviors [39]. The `NF-Crowd` protocols proposed in this paper minimize the fees for purchasing trust in crowdsourcing. On the other hand, it makes the off-chain workload heavier. In other words, the `NF-Crowd` protocols transform the mandatory charge of monetary fees into non-monetary off-chain workload, offering a new option to participants in crowdsourcing.

B. Decentralized crowdsourcing

Recent advancements in blockchain technologies [29] and smart contract platforms like Ethereum [36] are driving the rise of decentralized crowdsourcing systems [5], [14], [18], [26], [27], [34], [37], [38]. In contrast to centralized crowdsourcing platforms, decentralized crowdsourcing systems leverage the distributed miners to decentralize both data storage and computation in a tamper-resilient manner. Thus they eliminate the need for a trusted third party. However, existing designs of decentralized crowdsourcing systems can hardly handle transaction fees in a scalable manner, resulting in total costs of decentralizing crowdsourcing even higher than service fees charged by centralized crowdsourcing platforms. For instance, decentralized CrowdBC [26] charges 0.011 ether (i.e., \$1.93 by taking the average gas and ether prices in first half of year 2019) to tag 100 images while the same task only spends about \$0.45 in centralized AMT [32]. In [14], aggregating data from 1,000 data providers costs \$140. In [37], adding each task to the task matching smart contract cost about 210,000 gas (i.e., \$0.61), namely \$610 for adding 1,000 tasks. The `NF-Crowd` protocols proposed in this paper reduce the cost of running decentralized projects on top of Ethereum to a small constant value regardless of the scale of the crowd, which for the first time demonstrates a significant economic advantage in decentralizing crowdsourcing. In addition, we consider `NF-Crowd` a generic feature that is orthogonal to other design goals of decentralized crowdsourcing systems, allowing existing protocols to also become `NF-Crowd` by simply identifying TYPE $n \times 1$ and TYPE $1 \times n$ steps and reducing their cost with our strategies.

C. Scaling blockchain with off-chain execution

Off-chain execution of smart contracts is a promising solution for improving blockchain scalability [6], [9], [15]. However, recent works in this line have to either assume one honest manager for off-chain execution [6] or allow the execution to get aborted when the manager is dishonest [9]. The state channel network (SCN) [15] could achieve the never abort property when at least one participant is honest but it only supports two-participant contracts. The `NF-Crowd` protocols proposed in this paper extend the objective to support complex multi-participant multi-round smart contracts without

losing the never abort property. In addition, the NF−Crowd protocols for CC-OCR projects have been implemented in Ethereum, so they are ready-to-use.

D. Using cryptocurrency as security deposits

There have been many recent efforts on blockchain-based protocol design that leverage cryptocurrency as security deposits to penalize unexpected behaviors and improve security [2], [12], [24], [28]. In [12], ether is used as security deposits to provide verifiable cloud computing. In [28], ether is used as security deposits to enforce certificate authorities to be honest. Inspired by these previous efforts, NF−Crowd demands each participant to lock ether in smart contracts as security deposits to penalize potential misbehaviors violating the protocol and thereby enforces participants to stay honest.

VII. Conclusion

This paper proposes a new suite of protocols called NF−Crowd that reliably resolves the scalability issues faced by decentralized crowdsourcing projects. The proposed approach reduces the lower bound of the total cost to $O(1)$. We prove that as long as participants of a project powered by NF−Crowd are rational, the $O(1)$ lower bound of the cost could be reached regardless of the scale of the crowd. We also demonstrate that as long as at least one participant of a project powered by NF−Crowd is honest, the project cannot be aborted and the results are guaranteed to be correct. We design NF−Crowd protocols for a representative type of project named crowdsourcing contest with open community review (CC-OCR). We implement the protocols over the Ethereum official test network. Our results demonstrate that NF−Crowd protocols can reduce the cost of running a CC-OCR project to less than $2 regardless of the scale of the crowd, providing a significant cost benefit in adopting decentralized crowdsourcing solutions.

Acknowledgement

We thank our shepherd, Alysson Bessani and the anonymous reviewers for their feedback and comments. Chao Li acknowledges the partial support by Fundamental Research Funds for the Central Universities (No. 2019RC038).

Reference

[1] 99designs. https://99designs.com/, [Online].
[2] Marcin Andrychowicz et al. Secure multiparty computations on bitcoin. In *Security and Privacy*, pages 443–458. IEEE, 2014.
[3] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.
[4] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3:37, 2014.
[5] Diogo Calado et al. Tamper-proof incentive scheme for mobile crowdsensing systems. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–8. IEEE, 2018.
[6] Raymond Cheng et al. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *2019 IEEE European Symposium on Security and Privacy*, pages 185–200. IEEE, 2019.
[7] Anand Inasu Chittilappilly, Lei Chen, and Sihem Amer-Yahia. A survey of general-purpose crowdsourcing techniques. *IEEE Transactions on Knowledge and Data Engineering*, 28(9):2246–2266, 2016.
[8] UNOQLO Global T-Shirt Design Competitions. https://www.uniqlo.com/utgp/2020/hk/en/, [Online].
[9] Poulami Das et al. Fastkitten: Practical smart contracts on bitcoin. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019.
[10] Sourav Das, Vinay Joseph Ribeiro, and Abhijeet Anand. Yoda: Enabling computationally intensive contracts on blockchains with byzantine and selfish nodes. *NDSS*, 2019.
[11] designContest. https://www.designcontest.com, [Online].
[12] Changyu Dong et al. Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 211–227. ACM, 2017.
[13] John R Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer, 2002.
[14] Huayi Duan et al. Aggregating crowd wisdom via blockchain: A private, correct, and robust realization. In *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom2019)*, pages 43–52. IEEE, 2019.
[15] Stefan Dziembowski et al. General state channel networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 949–966. ACM, 2018.
[16] Stefan Dziembowski et al. Perun: Virtual payment hubs over cryptocurrencies. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 327–344, 2019.
[17] Etherscan: gas price. https://etherscan.io/chart/gasprice.
[18] Wei Feng and Zheng Yan. Mcs-chain: Decentralized and trustworthy mobile crowdsourcing based on blockchain. *Future Generation Computer Systems*, 95:649–666, 2019.
[19] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In *Annual International Cryptology Conference*, pages 291–323. Springer, 2017.
[20] Adam Groce and Jonathan Katz. Fair computation with rational players. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 81–98. Springer, 2012.
[21] Siyao Guo et al. Rational sumchecks. In *Theory of Cryptography Conference*, pages 319–351. Springer, 2016.
[22] LEGO Ideas. https://ideas.lego.com/, [Online].
[23] Upwork investor relations. https://investors.upwork.com/, [Online].
[24] Aggelos Kiayias and Qiang Tang. Traitor deterring schemes: Using bitcoin as collateral for digital content. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 231–242. ACM, 2015.
[25] Kovan. https://kovan.etherscan.io/.
[26] Ming Li et al. Crowdbc: A blockchain-based decentralized framework for crowdsourcing. *IEEE Transactions on Parallel and Distributed Systems*, 30(6):1251–1266, 2018.
[27] Yuan Lu, Qiang Tang, and Guiling Wang. Zebralancer: Private and anonymous crowdsourcing system atop open blockchain. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 853–865. IEEE, 2018.
[28] Stephanos Matsumoto and Raphael M Reischuk. Ikp: Turning a pki around with decentralized automated incentives. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 410–426. IEEE, 2017.
[29] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
[30] Thanh Hong Nguyen et al. Analyzing the effectiveness of adversary modeling in security games. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.
[31] The solidity contract-oriented programming language. https://github.com/ethereum/solidity.
[32] Amazon Mechanical Turk. https://www.mturk.com/, [Online].
[33] Upwork. https://www.upwork.com/, [Online].
[34] Jingzhong Wang et al. A blockchain based privacy-preserving incentive mechanism in crowdsensing applications. *IEEE Access*, 6:17545–17556, 2018.
[35] Whisper. https://github.com/ethereum/wiki/wiki/Whisper.
[36] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151, 2014.
[37] Yiming Wu, Shaohua Tang, Bowen Zhao, and Zhiniang Peng. Bptm: Blockchain-based privacy-preserving task matching in crowdsourcing. *IEEE Access*, 7:45605–45617, 2019.
[38] Xiaolong Xu et al. A blockchain-powered crowdsourcing method with privacy preservation in mobile environment. *IEEE Transactions on Computational Social Systems*, 2019.
[39] Yu Zhang and Mihaela Van der Schaar. Reputation-based incentive protocols in crowdsourcing applications. In *2012 Proceedings IEEE INFOCOM*, pages 2140–2148. IEEE, 2012.