CountChain: A Decentralized Oracle Network for Counting Systems

Anonymous Submission

***Abstract*—The integration of blockchain in industries like online advertising is hindered by its connectivity limitations to off-chain data. These industries heavily rely on counting systems with the goal of collecting and analyzing data that can be off-chain. This, in turn, requires mechanisms, often called oracles, to feed off-chain data into smart contracts. However, current oracle solutions are ill-suited for counting systems since the oracles do not know when to expect the data, posing a significant challenge.**

To address this, we present CountChain, a decentralized oracle network for counting systems. In CountChain, data is received by all oracle nodes, and any node can submit a proposition request. Each proposition contains enough data to evaluate the occurrence of an event. Only randomly selected nodes participate in a game to evaluate the truthfulness of each proposition by providing proof and some stake. Finally, the propositions with the outcome of True increments the counter in a smart contract. Thus, instead of a contract calling oracles for data, in CountChain, the oracles call a smart contract when the data is available. Furthermore, we present a formal analysis and experimental evaluation of the system's parameters on over half a million data points to obtain optimal system parameters. In such conditions, our game-theoretical analysis demonstrates that a Nash equilibrium exists wherein all rational parties participate with honesty.

***Index Terms*—Counting System, Blockchain, Decentralized Oracle, Smart Contract**

I. Introduction

Counting systems are mechanisms that keep track of the occurrence of specific events. A common example is compensating digital creators based on content impressions. These systems are essential for online advertising and data analytic, where numerous transactions transpire within a short period of time. However, conventional centralized counting system designs are prone to data manipulation, inaccuracy, single point of failure and trust issues. These challenges can be effectively mitigated by adopting a blockchain-based approach, which leverages smart contracts and Decentralized Oracle Networks (DONs) [1] [2]. Smart contracts, and blockchain in general, are subject to a significant limitation known as the *blockchain oracle problem*, namely, the smart contracts' limited ability to interact with external data sources (off-chain data). This limitation is one of the most critical barriers to overcome if smart contracts are to achieve mass adoption across various markets and use cases [2] [3] [4]. Current blockchain networks use trusted entities called oracles to access off-chain data. To fully address the oracle problem, a decentralized oracle solution is required to eliminate data manipulation, inaccuracy, and single point of failure. A Decentralized Oracle Network (DON) accomplishes this by bringing together numerous independent oracle nodes and multiple reliable data sources to establish decentralization from end to end.

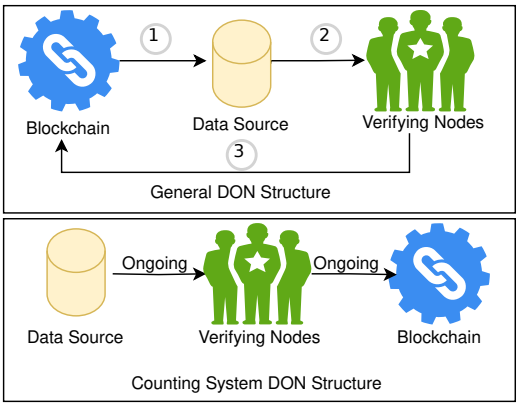

Fig. 1. General DON vs. counting system DON design.

Problem Statement: The current DON designs fail to provide a reliable solution for high-scale counting systems. This is a significant challenge to overcome if billion-dollar industries like online advertisement that rely heavily on counting systems are to adopt blockchain. In the current DONs, smart contracts call an outside source to gather off-chain data. The data is fed to multiple oracles in a DON to be verified. Once the trustworthiness of the data is validated, the data is fed back to the smart contract. Therefore, a smart contract initiates the process and knows when to expect the off-chain data.

In counting systems, events can be continuous and occur unpredictably at any moment. Therefore, the smart contract does not necessarily know when to expect the data. The data source collects the data and sends that data to the verifying oracle nodes in a DON. Once the DON validates the trustworthiness of the source and data, it triggers a condition in the smart contract. Then, the smart contract gets executed and receives the data. Figure 1 illustrates the difference between the existing DONs and the one needed for a counting system.

To put it into perspective, one can use ChainLink [3], one of the most advanced DONs, to aquire the current ETH price. To do so, the requester has to create a Service Level Agreement (SLA) contract to define the terms of the request (i.e., what is the price of ETH at a certain time). Then the network sends this request to some appropriate oracles. The oracles provide the requested data into an aggregating contract where the outcome is decided and reported to the requester. Then, the oracles receive some financial rewards.

This is not the case for counting systems. For instance, for counting the number of the website visitors that view an ad,

one can use Chainlink in three ways. One is to implement a SLA contract to request for the number of ad views in a certain period and the oracles respond with a number. Then, the Aggregating contract reports the aggregation of the results. The main problem with this approach is if there is a discrepancy between the oracles, it is challenging to resolve it. For instance, if three oracles report 10 views and two report 9, the network cannot decide if the two oracles missed counting the same ad view or different ones. The second approach is to implement SLA as a True/False proposition (i.e., whether user X viewed the ad on website Y). This approach is prone to single point of failure (the requester might miss some data) and is easy to cheat by the oracles if they are not required to provide any proof. The last approach is the combination of the two where the SLA contract requests for the number of ad views in a certain period and one oracle submits a proposition and the other oracles agree or disagree. This approach requires a new consensus protocol in addition to the current Chainlink protocol so that the oracles do not cheat or lazy vote.

Additionally, for all the cases, scalability can be a challenge and providing a financial reward for every request requires many transactions that increases the costs of the system. Also, if the event is ongoing, the SLA needs to be implemented as such which can increase the complicity of the design. Thus, in a nutshell, a smart contract requests data from the outside sources in the current designs while a counting system requires a design where the outside source calls a smart contract when the data is available.

Approach and Results: We propose CountChain, a DON designed for counting systems. All the blockchain nodes in the system receive the counting data from a source. The nodes can then have the role of a *submitter* or a *verifier*. The submitter pays a fee to submit boolean *propositions* to the system. These propositions contain enough information for verifiers to validate their correctness. Verifiers are chosen randomly by the network and verify the assigned propositions by voting True or False and providing proof and a fee as a stake. The majority of the assigned verifiers' votes decide the proposition's outcome. The nodes receive points for raising a proposition, or a valid vote for a True proposition. The accumulated points are then converted to financial rewards. Meanwhile, a dishonest vote/proposition results in the node losing money and points. The system is designed to encourage players to vote honestly. This system can be implemented separately or on top of the existing platforms like Chainlink.

We make three major contributions. First, we introduce CountChain, a DON designed for counting systems, offering enhanced trust, scalability, and resistance to common attacks. Secondly, through a game-theoretical analysis we demonstrate the existence of a Nash equilibrium where honesty is the optimal strategy. Lastly, our extensive experiments, involving over half a million data instances, demonstrate CountChain's feasibility and resilience to Sybil attacks. Together, these contributions introduces a reliable DON design for counting systems that can be used across various applications.

II. Related Work

The challenge of smart contracts accessing the off-chain data has spawned numerous DON proposals. Chainlink [3] aims to provide a cross-chain platform for internet-available data such as tamper-proof price data, automation functions, and external APIs. Neo Oracle Service [5] facilitates smart contracts to access external data by having oracle nodes designated by the committee fetch the data. Augur [4] and Gnosis [6] are prediction market platforms built upon Ethereum. TON [7] is a decentralized messaging application that relies on messages being fed to the chain and then verified by smaller agents. A Dag-Based Decentralized Oracle Model [8] is a decentralized oracle system that validates the input data, such as the current Bitcoin price, from a centralized source by using a decentralized directed acyclic graph mode. Distributed blockchain Price Oracle [9] is a distributed system that provides exchange rates of digital assets to smart contracts. This network is used as a safety for decentralized finance applications. Astraea [10] is a DON based on a voting game that decides the truth or falsity of propositions. Finally, A Smart Contract System for Decentralized Borda Count Voting [11] is a self-tallying decentralized e-voting protocol for a ranked-choice voting system based on Borda count.

With all these designs, decentralized solutions have yet to be presented for a counting system that can scale high. as mentioned in the problem statement part of Section I, one needs to make a significant effort to use these systems as a DON for counting systems.

III. Preliminaries

A. Decentralized Oracle Networks

Smart contracts' limited ability to interact with the off-chain data is a significant limitation which can prevent smart contracts from reaching about 90% of their potential use cases [1]. An intermediary system called an oracle is necessary to establish a connection between smart contracts and off-chain data. Decentralized Oracle Networks (DONs) are used to input off-chain data into a blockchain to combat these challenges. A DON offers a solution to single points of failure in smart contracts by employing multiple data sources. Doing so ensures end-to-end reliability and makes it possible for high-value smart contracts to operate in low-trust environments.

B. System Assumptions

In the design of CountChain, we made the following assumptions:

1) pseudo-random number generation is possible.
2) A proposition p has a truth value of either True or False.
3) There are N nodes numbered from 1 to N. For each p and each node $i \in [1, N]$, let $b_i(p) \in \{T, F\}$ denote the belief of i regarding the truth value of p. Each i has an accuracy $q_i(p) \in [0, 1]$ that is, informally, the probability that i is correct about their vote to a given proposition.
4) Each node's belief is independent of all others.
5) Honest players always vote on the assigned propositions and are truthful to the best of their knowledge.

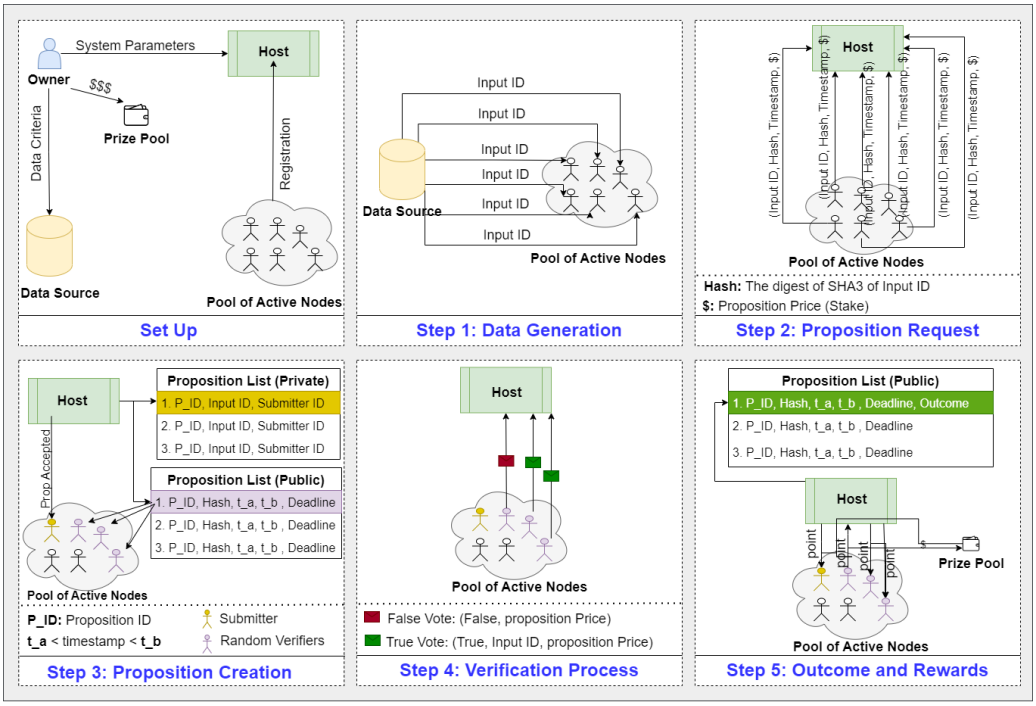

Fig. 2. An Overview of CountChain

Term	Definition	Set by
owner	the party interested in obtaining the result of the counting system	network
host	platform storing all information	owner/ network
player	participating nodes	network
∼ ID	unique identifier of each player	host
∼ point	the points accumulated for each vote or proposition submission	host
proposition	a boolean request to be validated	submitter
∼ deadline	the deadline to vote for a proposition	owner/ host
∼ pool	the list of available propositions to be verified	host
∼ price	the stake used when players vote for propositions	owner
submitter	the player who submits a proposition to be verified	players/ host
verifier	randomly chosen player to verify a proposition	host
input ID	unique identifier of the input data	data source
prize pool	financial reward and incentive	owner/ players

TABLE I
SUMMARY OF THE TERMINOLOGY.

IV. Description of CountChain

A. Setup and Terminology

The terminology summary is provided in Table I. This system requires an *owner* and a *host* to set the system parameters. The host is responsible for storing the data regarding *propositions* and the information of the nodes. The nodes (oracles) are just the validators of the system, similar to the miners in a blockchain network. The host is responsible for choosing a random selection process to ensure proper distribution of randomness across all nodes. The owner is the party interested in obtaining the result of the counting system. The owner provides the initial funds of the system and is responsible for defining the criteria for the counting object and other system parameters. The host and owner could be constructed by using a separate smart contract as the host and the contract owner or a different contract as the owner. The implementation of the host is outside the scope of this paper.

Each event or object being counted is assigned a unique identifier, known as the Input ID. The system is based on a set of *propositions* with covert truth values and associated price amounts, represented by *proposition price*, used in a voting game. Each proposition has a termination condition by which the proposition outcome is decided. The owner sets this termination condition. Generally, the termination condition is the *proposition deadline*, a deadline by which all the votes need to be submitted. The system's fund or the *prize pool* is the accumulation of the initial fund provided by the owner and the stakes collected from the dishonest Players. The propositions are stored in the propositions Pool.

Each Player (node) in the system is given two entities of *player ID* and *player point*. The player ID is a unique identification of the Player (i.e., its blockchain address), and the player point is the points accumulated by the player for submitting a Vote or proposition. The final reward of each Player is decided based on the player points. Finally, each Player can have two roles in this system: the *submitter* that submits a proposition to be verified by the system, and the *verifier* that verifies the validity of each assigned proposition.

Algorithm 1: Pseudocode for the Host

```
propositions ← []
def PropositionSubmission(request):
    if InvalidHash(request) then
        return WrongHash(request)
    end
    if NOTPropositionExists(request) then
        T_a, T_b ← GetTimeSpan(request[TimeStamp])
        proposition ← CreateProposition(request, T_a, T_b)
        assignedVerifiers ← AssigneRanomVerifier(request)
        for verifier in assignedVerifiers do
            SendProposition(verifier, proposition)
        end
    end
end
def VoteSubmission(voteReq):
    if NOTPassedDeadline(voteReq) then
        if voteReq[vote] == True AND InvalidHash(voteReq) then
            return WrongHash(voteReq)
        end
        AddVoteToProposition(voteReq[vote])
    end
end
```

Algorithm 2: Pseudocode for Verifiers

```
storedData ← []
//Listening for input from the Host or the Data Source
while Active do
    if data is Recieved then
        if data From Source AND data is Valid then
            storedData.add(data)
            hash ← ComputeHash(data[InputID])
            SendPropositionRequest(data, hash, TimeStamp)
        end
        if data From Host then
            vote, inputID ← False, NULL
            for entry in storedData | data[T_a] ≤ entry[time] ≤ data[T_b] do
                if ComputeHash(entry[InputID]) == data[hash] then
                    vote, inputID ← True, entry[InputID]
                    break
                end
            end
            SendVote(vote, inputID)
        end
    end
end
```

Fig. 3. Pseudocode for the Host and Verifiers

B. System Description

The data source send the data regarding the object being counted to all the active Players. Each Player stores this data independently. The Players know the criteria for the counting object ahead of time. For instance, if the system counts the number of visitors to a particular website, the counting criteria is the number of users that successfully load the website. The visitor's IP address is the Input ID, and the Players can receive the required information using invisible pixels.

Once the Players observe a valid event, they submit a proposition request to the host and pay the proposition price as a stake. The proposition request should contain the Input ID, Timestamp, player ID and Hash(Input ID) which is the digest of the cryptographic hash function of the Input ID.

Once the host receives a valid proposition request, it generates a proposition with a new format and adds it to the *proposition pool*. Then, the host assigns the proposition to some random Players. These selected Players become the verifiers of the proposition. The generated proposition by the host does not contain the Input ID; instead, it contains the Hash(Input ID). Also, the Timestamp is converted to T_a and T_b, where the Timestamp is the midpoint of T_a and T_b. This is because the Players may receive the information at different times due to delays. The proposition also contains the proposition deadline, the time the assigned verifiers must submit their votes. The Player that submitted the proposition request becomes the submitter of the proposition. The propositions are publicly available to all Players, but only the assigned verifiers can vote. The number of assigned verifiers for each proposition is decided by the owner and is fixed for all the propositions.

In short, the propositions are public and contain Hash(Input ID), T_a, T_b, proposition deadline, a list of assigned verifiers bit, not the Input ID. The verifiers only need to keep the received data for a specific amount of time. Once the outcome is decided and added to the blockchain, verifiers can delete the data on their end.

Verification Process: The assigned verifiers calculate the hash of all the Input IDs in the period of T_a and T_b. If they find the corresponding hash as the Hash(Input ID) in the proposition, they vote True and submit the Input ID; otherwise, they vote False. If the assigned verifiers do not submit their votes by the deadline, it is regarded as voting False. The assigned verifiers must also pay the proposition price as a stake. Using hash functions make it difficult for the verifiers to guess the correct Input ID given the Hash(Input ID) if they do not have the correct information in their records. On the other hand, it is relatively cheap to find the corresponding Input ID from the limited received data. Therefore, when they vote True, they provide the User ID as proof of honesty.

The majority of the votes decide the outcome of the proposition. The verifiers in the majority get rewarded, and those in the minority get penalized. To break the tie, if the owner requires an even number of verifiers for each proposition, the submitter's vote is counted as a True vote.

Choosing a fixed random number of verifiers makes all propositions get an equal number of verifiers; hence, the results have similar accuracy. It also reduces the risk of sybil attacks by ensuring players do not team up to pick one proposition to vote. The overview of CountChain is shown in Figure 2, and Figure 3 shows the algorithms for steps 2 to 5.

C. Reward and Penalty

Whenever a Player submits a proposition request or votes True, the host verifies that the provided Hash(Input ID) and ID correspond to another. If incorrect, the Player loses the stake and receives two negative points for intentionally submitting a wrong format. This makes security attacks such as Denial of Service (DoS) expensive.

If most verifiers vote True for a proposition, the submitter receives four positive points, the verifiers that voted True receive one positive point, and the verifiers that voted False lose their stake and receive one negative point. If most verifiers vote False for a proposition, the submitter receives two negative points and loses the stake, and the verifiers that voted True

		Majority: True	Majority: False
Verifiers	True	+1pt	0
	False	−1pt, $-pp$	0
Submitter		+4pts	−2pts, $-pp$

TABLE II
POSSIBLE OUTCOMES OF PROPOSITIONS. HERE PT(S) STANDS FOR POINT(S) AND *pp* IS THE PROPOSITION PRICE.

or False do not get any points. Therefore, submitters take a higher risk and receive a higher reward for submitting a correct proposition. This is because one of the main factors that determine the quality of the network is the quality of the proposition requests submitted.

The points are added to the player points. Honest players get their stake back. The stake taken away from the dishonest Players is added to the prize pool. If the player point of a Player becomes lower than a threshold, they get banned from the network. Table II summarizes possible outcomes.

According to Table II, submitters need to submit the proposition request they genuinely believe is correct. Otherwise, they lose money and point. The reward for a True proposition is +4 points, and the penalty for submitting a False proposition or providing the wrong data is -2 points and losing the stake (proposition price). Hence, the best strategy is to submit a proposition request they believe is True.

Players receive a financial reward after a termination condition is met. This condition can bean be a fixed period of time, a number of propositions, or a certain number of Points. If the player point of a node is over zero, they get a financial reward based on their accumulated points using the below formula. In this formula p_{v_i} is the player point of a given Player, ($u(V_j)$ is a Heaviside function and $\sum_{j=1} p_{v_j} u(V_j)$ is the summation of the player points of all the Players with positive points

$$\text{Verifier Prize} = \text{Prize Pool} \cdot \frac{p_{v_i}}{\sum_{j=1} p_{v_j} u(V_j)}$$

The point of each Player gets reset after they are paid. There are several benefits of providing a financial reward based on a point system whereas paying after each proposition result. First, it motivates the verifiers to partake in the game continuously. It also incentivizes continuous honest behavior. If a verifier loses some points, they put more effort into the next propositions, and if a verifier gains some points, they are motivated to continue the honest behavior. It reduces the number of monetary transactions in the system and helps prevent lazy voting and reduces Sybil attacks.

D. DoS and Sybil Attack Prevention

In the case of a DON, a DoS attack can prevent a new proposition from being added. It can also prevent the verifiers from accessing a proposition, or some votes can be lost. In CountChain, only the randomly selected verifiers can vote on a proposition. The number of verifiers is based on the availability of the network. Therefore, the only option for an adversary is to use the proposition submitting mechanism to perform DoS attacks. When submitting a proposition request, the submitters are required to follow a specific format and perform some computations. If this requirement is not followed, the submitters lose points and money. After losing some points, the Player gets banned from the network. Hence, even in this case, a DoS attack is very costly. Thus, the design of CountChain minimizes the risk of DoS attacks.

A Sybil Attack is when an entity has numerous fake identities for malicious motives [12]. PoW, PoS, and DPoS are the main sybil control mechanisms used in Blockchain [13]. PoW safeguards against Sybil attacks by requiring nodes to employ a limited resource to create new blocks. Hence, creating numerous nodes becomes ineffective. [14] [15]. However, this design has a high energy consumption [16].

PoS addresses the drawback of PoW by assigning voting power based on the number of tokens locked to a node. Therefore, attacking the network becomes costly, as an attacker must gather a significant portion of the total staked tokens, potentially risking a substantial capital investment [15]. However, the more stakes a node has, the more blocks it can create. Thus making it more powerful. Since the number of coins/tokens is limited, once a node gains 51% of the total stake, it achieves the majority of the power that cannot be revoked later [16].

DPoS is a type of PoS where token holders can delegate their tokens to node-runners, which increases the tokens required for a Sybil attacker. The main limitation of this system is that it can become centralized more easily when there are fewer delegates, as a malicious coalition of at least 51% of delegates could control the network more easily. This is a significant issue as most projects allocate significant tokens to Venture Capitals and insiders [17].

CountChain minimizes sybil attacks using system parameters that are adjustable by the host based on the needs. Also, few verifiers are randomly chosen from a pool of available verifiers, which along with the point and stake system, makes sybil attacks more expensive. Finally, an authentication step can be added for each node.

E. Limitations of CountChain

CountChain is a DON aimed at counting systems or any system that requires a precise counter, and it does not apply to most other applications. Moreover, like most other blockchain or decentralized applications, the quality of the network relies on the diversity and the number of available nodes. Finally, CountChain requires an owner to determine some parameters. This limitation can be removed if, instead of an owner, the majority of the participants, a bigger blockchain like Ethereum or a smart contract sets these parameters.

V. GAME THEORY: THE VERIFIER'S DILEMMA

Lazy voting and Nash Equilibrium (NE) are two essential concepts when designing a DON. Lazy voting is when players blindly vote True or False, knowing either option is more likely to win. This increases dishonest voting and can significantly make the outcome inaccurate. Therefore, a DON needs to be designed to prevent lazy voting.

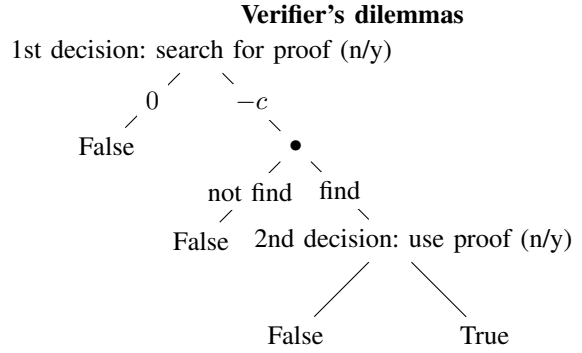

Fig. 4. The two verifier's dilemmas in extensive form. A verifier needs to make two decisions: whether to put effort with sunk cost $-c$ and search for proof and if they find proof, whether to use it and vote "True" or ignore it and vote "False".

		Majority: True	Majority: False
Verifier	True	[1, 0], [1, 0]	[0, 0], [0, 0]
	False	[−1, −1], [1, 0]	[0, 0], [0, 0]

TABLE III
SUMMARY OF PAYOFFS. THE ENTRIES IN EACH BRACKET CORRESPOND TO [POINT, STAKE].

In CountChain, lazy voting True is not possible as verifiers must provide proof when voting True. On the other hand, the best reward for voting False is not to get penalized. This means lazy voting False has a high risk (losing money and points) and no reward. Therefore, verifiers are only incentivized to vote False if they perform the computation and ensure that the correct vote is False. Also, not voting is considered voting False. This ensures inactive players are not incentivized to stay in the game. Moreover, submitting a False proposition request gets financial and point punishment; hence, submitters need to put in effort to ensure their submission is valid.

A Nash equilibrium (NE) is a situation in game theory where no player can benefit by changing their strategy, given all other players' strategies remain the same. Analyzing NE ensures that the DON design incentivizes honest behavior.

In CountChain's design, in each round, every verifier needs to make two decisions. These are depicted in Figure 4. The first decision is whether to search for proof or not. Searching for proof costs $c > 0$ for the verifier, i.e., the verifier earns $-c$ from doing the search, whereas doing nothing incurs a cost of 0 where c is the computational costs. If a verifier does not search for a proof, then they may only vote "False" according to the rules of the protocol. The same holds in case that they search for a proof, but do not find one. However, if they find a proof, then they are faced with the second dilemma which is whether to use this proof and vote "True" or ignore the proof and vote "False".

The utility of a verifier depends on its decision and the behavior of other verifiers. The summary of the payoffs is shown in Table III. From Table III, it is immediate that first, it is optimal for each verifier to be in the majority and second, that it is a weakly dominant strategy for a verifier to vote True whenever they possess a valid proof. Specifically, if the majority votes True, then the verifier is strictly better to vote True than False and if the majority votes False, then the verifier is indifferent between the two.

The game between verifiers may possess many Nash equilibria. For instance, if all (other) verifiers engage in lazy voting, then it is best for a remaining verifier to do nothing and vote False by default. The reason is that if every other verifier is voting False, then no proposition will be decided as True, and incurring the cost of searching for proofs will only result in a negative utility for an honest verifier.

In view of the above, the main question that we need to answer is whether honest behavior, i.e., searching for proofs and submitting them (voting True) if one finds them, is a Nash equilibrium in the game. Assuming a fraction $p_T \in (0, 1]$ of True propositions, this is answered affirmatively in Theorem 1 under the mild assumption that $2p_T > c$, where c denotes the cost of searching for a valid proof (typically assumed to be very low since it only involves the calculation of a small number of hashes).

Theorem 1 (Incentive Compatibility of CountChain)**.** *Assume that a fraction $p_T \in (0, 1]$ of propositions are True and that searching for a valid proof incurs a cost $c > 0$ to the verifier. Then, if all other verifiers are behaving honestly, i.e., search for valid proofs and vote True whenever they find one, then it is best for a remaining verifier to also behave honestly if and only if $2p_T > c$.*

Proof. Assuming that all verifiers behave honestly ensures that a fraction of p_T propositions will be decided as True, i.e., all True propositions will be decided as True and all False propositions will be decided as False. Thus, according to Table III, the expected utilities of a single verifier from their possible actions $\{no{-}search, search{-}false, search{-}true\}$ are the following

$$\begin{aligned}
\mathbb{E}[\text{no-search}] &= p_T(-1) + (1 - p_T) \cdot 0 = -p_T \\
\mathbb{E}[\text{search-false}] &= -c + p_T(-1) + (1 - p_T) \cdot 0 = -p_T - c \\
\mathbb{E}[\text{search-true}] &= -c + p_T \cdot 1 + (1 - p_T) \cdot 0 = p_T - c
\end{aligned}$$

Clearly, the action search-false is dominated by no-search, since $-p_T - c < -p_T$ for $c > 0$. Thus, to determine a verifier's optimal action, we only need to consider the actions "search" and "vote True" whenever one possesses a valid proof. The expected utility of a verifier who selects "search-true" $x \in [0, 1]$ fraction of time and "no-search" $1-x$ fraction of time is

$$\begin{aligned}
\mathbb{E}[x] &= x\left[-c + p_T \cdot 1 + (1 - p_T) \cdot 0\right] \\
&\quad + (1 - x)\left[p_T \cdot (-1) + (1 - p_T) \cdot 0\right] \\
&= x(2p_T - c) - p_T.
\end{aligned}$$

This is increasing in x whenever $2p_T > c$ as claimed. In particular, the verifier's utility is optimized for $x = 1$ which corresponds to always honest behavior. □

The assumption of Theorem 1 that all other verifiers behave honestly is only needed to ensure that the correct fraction of

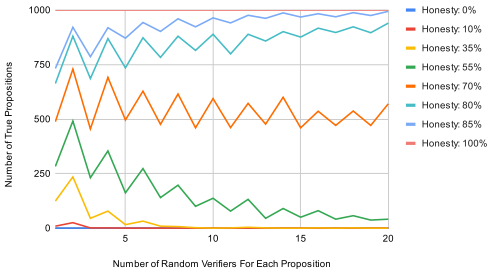

Fig. 5. Optimal Honesty Rate and the Number of Verifiers

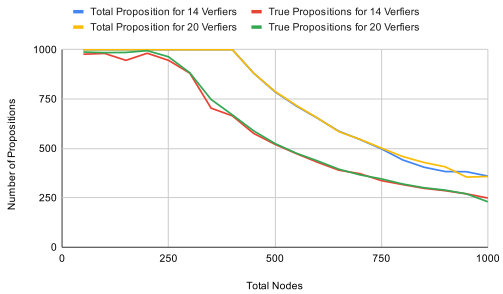

Fig. 6. Optimal Number of Nodes

True propositions is decided as True. This assumption can be relaxed to a *(super) majority* of verifier's behaving honestly, while still maintaining that a single verifier is better off by also behaving honestly.

VI. Experimental Evaluation

A. Preliminary and Experimental Setup

As stated in section II, the available designs require a major change to be used for counting systems. CountChain is created to fill a pervasive need for a practical DON for counting systems. As a result, we performed multiple experiments to test the feasibility of the design of CountChain and determine the optimal system parameters. The experiments were run on an Azure E96ds v5 virtual machine equipped with 96 VCPUs and 672 GiB memory on Ubuntu 20.04 OS. The source code and the results can be found in here.[1]

B. Optimal System Parameters

This experiment aims to demonstrate the impact of system parameters. These parameters are each node's honesty rate, the number of chosen random verifiers for each proposition and the total number of nodes. Honesty rate mimics the rate of accuracy or honesty of a node, as nodes may intentionally or unintentionally vote incorrectly for a proposition.

In this experiment, first, we altered the honesty rate for each node from 0% to 100% in increments of 5% means "not honest" and 100 % means "most honest". We changed the number of random verifiers assigned to each proposition from 1 to 20 with increments of 1 for each honesty rate. Each time, we generated and sent 1,000 random valid input data for a total of 420,000 valid data at the rate of 10 data per second. In the best case, we expected to see 1,000 propositions with the True outcome for each scenario; hence, we used these values as the ground truth. The verifiers were picked at random from a pool of 100 available nodes. We used the following values for other parameters: proposition price :1, proposition deadline: 2 sec, proposition Delay: 1 sec, and Threshold Point: -5.

The summary of the results are shown in Figure 5. As expected, the best outcome was when the honesty rate of all nodes was 100%, whereas the worst result occurred when the rate was 0%. Also, when the rate was 10% or higher, all the propositions were successfully raised, and in the case of 5%, only a small number of propositions were not submitted. This is because all nodes have access to the data and can initiate a proposition request. Therefore, this result has not been included in the charts above to avoid duplicity.

The first noticeable accuracy loss becomes evident between 85% and 80%. This is because as the honesty rate of all nodes decreases, the chance of the majority being on the inaccurate side increases. As illustrated, a precision level of up to 99% can be achieved with the honesty rate of 85%, whereas the maximum accuracy attainable is 95% for the rate of 80%. Hence, it is recommended to configure the system to ensure all nodes possess an honesty rate of at least 85%. This is by filtering the nodes proven to be unreliable. Additionally, for this honesty rate, the highest level of precision was observed for 20, 18, and 14 verifiers, resulting in 994, 988, and 987 True propositions, respectively. Thus, the recommended optimal number of chosen verifiers per proposition is fourteen, which reasonably balances accuracy and resource allocation.

Furthermore, for the honesty rate of over 80%, increasing the number of verifiers improves the precision of the outcomes. This is the opposite when the honesty rate is below 60%. This is because most verifiers are honest if the honesty rate is high for a given proposition. Also, when all nodes' honesty rate declines below 65%, having only two verifiers results in substantially more accurate outcomes. These observations highlights that the accuracy of the results is significantly influenced by the assignment of the system's parameters and the nodes' quality can significantly impact decentralization. Additionally, an even number of verifiers yields higher accuracy. This is because the submitter's vote is only considered and counted as True in the event of a tie in the overall vote, which can only occur when the number of verifiers is even. Nevertheless, this observation highlights that this rule improves the precision of the system's outcomes.

Then, we varied the number of total nodes from 50 to 1,000 in increments of 50. fourteen verifiers are randomly picked for each proposition from the pool of nodes. The honesty rate for all the nodes is set to 85%. We also repeated the experiment by picking 20 verifiers to compare the results.

As shown in Figure 6, increasing the number of nodes from 50 to up to 250 can improve the accuracy in both cases.

[1]https://anonymous.4open.science/r/CountChain-058D/README.md

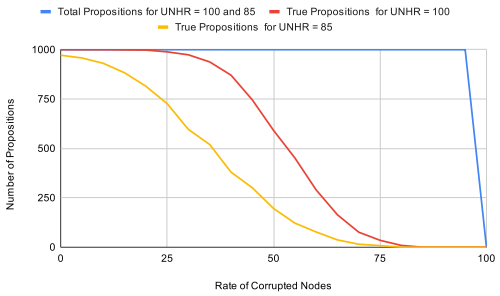

Fig. 7. Sybil Attack Success Rate. UNHR = uncorrupted node honesty rate

However, the performance starts to decrease when there are more than 250 nodes. When the total number of nodes is more than 400, the number of raised propositions decreases as the number of nodes increases. Also, when the total number of nodes is more than 250, the number of True propositions decreases as the number of nodes increases.

Our performance analysis shows that every extra 50 nodes results in an extra 0.6 to 1 % CPU and RAM usage and no noticeable disk usage. However, the main bottleneck is the allocated heap memory for the program. Even when set to max, the program reaches the maximum allocation when the number of nodes is high. Therefore, this is mainly a limitation of the available hardware resources than the design. Hence, this limitation should be handled if a distributed infrastructure is used as needed for a blockchain design.

Moreover, the vulnerability of a system to DoS attacks increases with a higher number of nodes, while a lower number of nodes increases the chances of successful Sybil attacks. Therefore, finding the optimal number of total nodes is vital based on the system's infrastructure and the application's requirements. In our setup, 200 nodes yield the most favorable results. Furthermore, when fourteen verifiers were chosen from a pool of 200 nodes, 982 propositions out of the raised 1,000 were correctly decided as True. Meanwhile, this number is 995 when the number of verifiers was 20. This indicates a less than 1% increase in accuracy for using six additional verifiers per proposition. These results demonstrate that the optimal balance between accuracy and resource allocation is achieved with fourteen verifiers from two hundred nodes.

C. Sybil Attack Success Rate

This experiment aims to simulate a Sybil attack and find the success rate of this attack on CountChain. In this experiment, fourteen verifiers were picked from the pool of two hundred nodes. A Sybil attack is considered entirely successful if disrupting nodes prevents proposition creation or they incorrectly make the outcome of all the generated propositions False.

In this experiment, we set the honesty rate of the corrupted nodes to 0% and varied the rate of corrupted nodes from 0 to 100% (in increments of 5). Generally, when determining the success rate of a Sybil attack, the honesty of all the honest nodes is considered 100%. Therefore, we decided to have two sets of test cases with the honesty rate of honest nodes as 85% and 100%. For example, if there are 200 nodes where half are corrupted, 100 random nodes would have an honesty rate of 0, and the honesty rate for the rest would be 85% and 100%. The rest of the parameters are similar to previous experiment.

As shown in Figure 7, when the honesty rate of uncorrupted nodes was 100%, even when the corrupted nodes gained 50% of the network, over half of the propositions were correctly decided as True. The attack was not entirely successful until corrupted nodes gained at least 85% of the network. Meanwhile, changing the honesty rate of uncorrupted nodes to 85% significantly increased the attack's success rate. In this case, the attackers needed to gain 70% of the network to launch the attack successfully. When the attackers gained 50% of the network, only about 20% of the propositions were correctly decided as True, indicating a partial success of the attack. These results, once again, demonstrate that nodes' quality significantly impacts the network's performance. Another notable observation is that in both cases, all the propositions were successfully raised until the corrupted nodes completely controlled the network. This is because all the data is sent to all the nodes, and as long as one honest node is active, they can raise the proposition. Although this does not prevent the attack, it can be used as reference data for the future.

This experiment can also be performed differently, where an adversary generates a fake proposition and distributes the Input ID to the other colluded nodes to vote True if they are picked to verify that proposition. Unlike the previous case that the verifiers that vote False are at risk; in this case, the submitter takes more risk. This is because if the majority votes False, the submitter loses stake and receives some negative points. Therefore, after accumulating some negative points, the dishonest nodes get banned. As a result, the outcome of this case is similar to the case shown in this experiment.

The results show that CountChain performs better than other similar designs due to multiple factors included in the design. First, only a selected number of nodes can vote on a proposition. In most other DONs, all nodes can participate freely, so if an entity achieves the majority of the power in a network, they are guaranteed to succeed in the attack. In CountChain, the attackers need to gain more than the majority for guaranteed success, as shown in this experiment. Also, the point and stake system increases the cost of DoS and Sybil attacks. Finally, nodes accumulating a few negative points are banned from the system, making a Sybil attack harder.

VII. Conclusions and Future Work

One of the main barriers to the mass adoption of blockchain is its limitation to interact with off-chain data. This paper summarizes the design and implementation of CountChain, a blockchain oracle for counting systems. This opens many doors to billion-dollar industries like online advertisement that rely on counting systems to adopt blockchain. In the future, we plan to expand our design to an industry standard platform for commercial use as it possesses tangible commercial potential that can improve the decentralized ecosystems.

REFERENCES

[1] Chainlink, "What is the blockchain oracle problem?" 2020. [Online]. Available: https://blog.chain.link/what-is-the-Blockchain-oracle-problem/

[2] G. Greenspan, "Why many smart contract use cases are simply impossible," 2023. [Online]. Available: https://www.coindesk.com/markets/2016/04/17/why-many-smart-contract-use-cases-are-simply-impossible/

[3] S. Ellis, A. Juels, and S. Nazarov, "Chainlink a decentralized oracle network," 2017. [Online]. Available: https://research.chain.link/whitepaper-v1.pdf

[4] J. Peterson, J. Krug, M. Zoltu, A. K. Williams, and S. Alexander, "Augur: a decentralized oracle and prediction market platform (v2.0)," 2022. [Online]. Available: hhttp://www.augur.net/whitepaper.pdf

[5] D. HongFei and E. Zhang, "Neo oracle service," 2014. [Online]. Available: https://docs.neo.org/docs/en-us/advanced/oracle.html

[6] Gnosis, "Gnosis whitepaper," 2017. [Online]. Available: https://www.allcryptowhitepapers.com/gnosis-whitepaper/

[7] N. Durov, "The open network," 2021. [Online]. Available: https://ton.org/whitepaper.pdf

[8] R. Gouiaa, F. Hdhili, and M. Jansen, "A dag based decentralized oracle model: Implementation and evaluation," 2021. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-86162-9_31

[9] L. Lys and M. Potop-Butucaru, "Distributed blockchain price oracle," 2022. [Online]. Available: https://eprint.iacr.org/2022/603

[10] J. Adler, R. Berryhill, A. Veneris, Z. Poulos, N. Veira, and A. Kastania, "Astraea: A decentralized blockchain oracle," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018, pp. 1145–1152.

[11] S. Panja, S. Bag, F. Hao, and B. Roy, "A smart contract system for decentralized borda count voting," 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9089037

[12] J. Fáwolé and L. Ciattaglia, "Sybil attack in blockchain: Examples & prevention," 2023. [Online]. Available: https://hacken.io/insights/sybil-attacks/

[13] J. Che, Y. Zhao, and X. Chen, "Oscms: A decentralized open-source coordination management system using a novel triple-blockchain architecture," 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/11/6580

[14] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[15] RADIX, "What are proof of work and proof of stake?" 2023. [Online]. Available: https://learn.radixdlt.com/article/what-are-proof-of-work-and-proof-of-stake

[16] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and Cryptocurrency Technologies*. Princeton University Press, 2016.

[17] Crypto.com, "What is delegated proof of stake?" 2022. [Online]. Available: https://crypto.com/university/what-is-dpos-delegated-proof-of-stake