MnemoSys: A Conditional Probability Estimation Protocol for Blockchain Audited Reputation Management

1st Daniel Rouhana*
Department of Computer Science
Washington State University
daniel.rouhana@wsu.edu

2nd Peyton Lundquist*
Department of Computer Science
Boise State University
peytonlundquist@u.boisestate.edu

3rd Tim Andersen
Department of Computer Science
Boise State University
tandersen@boisestate.edu

4th Gaby G. Dagher
Department of Computer Science
Boise State University
gabydagher@boisestate.edu

***Abstract*—Reputation systems have been one method of solving the unique challenges that face distributed networks of independent operators. Fundamentally, historical performance must be considered in a way that attempts to predict future behavior, optimize present functionality, and provide some measure of immutable recording. In this paper, a three-part system, MnemoSys, is proposed to solve this diverse set of problems. First, historical performance is dynamically weighted and scored using geometrically expanding time windows. Second, a quorum is abstracted as a restricted Boltzmann machine to produce a conditional probability estimate of log-normal likelihood of good-faith behavior. Third, all rewards and punishments are recorded on an immutable, decentralized ledger. Our experimentation shows that when applied iteratively to an entire network, consistently underperforming nodes are removed, network stability is maintained even with high percentages of simulated error, and global network parameters are optimized in the long-term.**

***Index Terms*—Blockchain; Reputation System; Boltzmann; Quorum; Distributed Systems**

I. INTRODUCTION

Distributed networks of independently operating computers require resilient protocols ensuring procedural integrity, validating data provenance, and proving accuracy. A potential solution to this problem is a distributed peer-to-peer network of nodes who rely on each other for computational validation; however, a robust system for measuring reputation is necessary for a dependable system. Each pre-existing implementation has its own features targeted for optimized performance, but the main requirements and challenges remain universal. Primarily, a hard measure of "trust" must be produced by the system as an operational metric for quantifying reliability of an operator or element in that system. The method for producing this value must have some dynamic capabilities to respond to changing network conditions, detect anomalous behavior, and identify likely malicious nodes. Specifically, nodes can operate in a few different ways that propose challenges to these systems. For example, a malicious node acting alone can alter its behavior after a period of high reputation. When acting together, malicious nodes can perform shilling attacks, where dishonest feedback is submitted to either boost the ratings of other malicious nodes or to lower ratings of non-malicious nodes.

Broadly speaking, there are two main avenues of approaching the above distributed network problems – structural and mathematical solutions. Structural solutions are usually mechanistic in nature, opting to architect transfer of data (rather than manipulation of the data) or gate network access in some capacity; these solutions can easily become contrived, difficult to scale, and often require varying degrees of centralization to function. On the other hand, mathematical solutions seek to detail a mechanism that strikes a balance between formal rigidity and ability to adapt to dynamically changing network conditions. For example, TrustGuard works to detect dishonest feedback using personalized trust values. Consequently, the trust value of a node from the perspective of other nodes may be very high, while its trust seen by the rest of the system might be low [1]. EMLTrust builds off of their work, instead using a biased support vector machine (SVM) classifier with pre-determined features [2]. This is an unworkable approach for decentralized systems, as it requires either a local copy of the SVM or some central processing station. EigenTrust determines reputation purely off of a matrix analysis of global reputation of a peer, given by local trust values weighted by global reputations of assigning peers [3]. To implement this in a blockchain-based reputation system would require an image of the global reputation values of the network, combinatorically expanded to hold every nodes' rating of every other node. However, the largest shortcoming of the above works is the assignment of reputation scores from other nodes, leaving the door open for shilling attacks from coordinated malicious nodes.

*These authors contributed equally.

With these considerations in mind, this paper presents the following contributions.

1) We propose a novel distributed system, named MnemoSys, that ensures integrity and accuracy of computations by keeping track of the reputation score of each user in the system using a decentralized ledger to record reputation adjustments.
2) MnemoSys uses dynamic historical performance scoring where time is sliced into geometrically expanding intervals, assigning an inverted harmonic weight to the mean performance within those time intervals to calculate a reputation score.
3) MnemoSys utilizes the Slow Boltzmann Estimator (SBE), a novel method of computing a probabilistic measure of confidence using the reputations of a stochastically selected quorum of users. A log-normal likelihood of good-faith behavior is produced, invariant of the performance of the current quorum, to then increment or decrement their respective reputation histories.
4) We implemented MnemoSys over a simulated distributed network of independent nodes parameterized to accurately simulate real-world performance. Our experimental evaluation results show that our system enforces honest behaviour by removing consistently under-performing nodes, accurately and immutably recording reputations, and stabilizes the network as a whole in conditions of extreme failure.

II. Related Work

In this section, we identify similar research in decentralized networks using reputation systems. Afterwards, we dive further in to research surrounding our abstraction of a Restricted Boltzmann Machine (RBM).

Research has been widely conducted for distributed networks using reputation mechanics, which associate participating nodes with a reputation score. This score allows many systems to make predictive decisions regarding a node's role within the network to facilitate more honest membership, as explored in [2][1]. Much research has focused on blockchain-based, distributed reputation systems, which store reputation on the blockchain, discussing these mechanisms [4][5][6][7][8][9]. Other papers [10][11] consider using blockchain based reputation systems in addition to Federated Learning, in order to produce a trustable ML model. Our proposed system relates to each of these categories of research, and explores mechanisms across each to deliver a blockchain based approach that can produce verifiable computation.

TrustGuard [1], a framework for efficient and dependable reputation systems, proposes a methodology to improve calculating a trust score beyond common simple averages. Specifically, their "Strategic Oscillation Guard" aims to prevent malicious nodes intelligently adapting behavior to game the system. A bad node may act in good faith, build a solid reputation, and then began acting maliciously. Alternately, a bad node may oscillate between good and bad behaviour in such away that the bad behaviour is sustainable. A concept described as "Fading Memories" is introduced, where reputation history of a node is weighted differently according to how distant in time the event occurred, with most distant events being weighted the least. Time events are then aggregated in to intervals, with further events being more aggregated; windows are then moved forward one step to produce an estimated confidence for the present time. We modified this concept in order to increase fairness to the system where nodes may be afforded forgiveness over time. This approach avoids punishing well-intentioned nodes who may have failed to perform their expected behaviour, for example by losing power during a single computation. The node would be punished in the short term, but forgiven if proved to be generally an honest actor.

Kaci and Rachedi [4] propose their PoolCoin blockchain wherein the system aims to solve the problem of dishonest miners joining and damaging mining pools. Their solution is to assign 'trust scores' to miners and managers, which are stored on the blockchain within transactions, providing an immutable record of reputation for each node in the system. From this trust score, they derive a cost the miner needs to pay to the mining pool manager in order to participate. If a miner submits valid PPoWs (Partial Proof-of-Works) to the Pool Manager, their trust score is increased, thus decreasing the cost of participation for the next round. In addition, PoolCoin offers a machine learning module that allows the Pool Manager to predict the genuine computational capacity of miners as an additional layer of honesty enforcement. In contrast to their solution, ours does not incorporate any monetary mechanisms; rather, we can adhere simply to the reputation principle. Their trust score limits the miners' participation by adjusting the cost to join. Our nodes' trust score directly influences whether members accept or reject the computation a miner performs. Leaning in to the reputation structure rather than on monetary mechanisms, will deter many malicious actors from organizing an attack.

A solution for a scalable BFT consensus for blockchain networks proposed by Bugday et al. [5] introduces a mechanism which selects a subset of nodes to participate in a consensus group for the network operations. This quorum is derived from an adaptive hedge method (AHM) that uses the on-chain reputation values. This employs a triangular distribution across the network's nodes as to allow some low confidence nodes to enter the quorum for fairness. Similarly, MnemoSys implements a quorum-based consensus which favors quorums made of reputable nodes. Although, rather than using the AHM, we choose the Boltzmann machine to predict quality quorums, but regardless of the quorum's reputation composition, the group is still allowed to operate. However, participation does not guarantee their consensus is valid, but it may still positively impact the nodes' reputations. This unique mechanism provides a maximum fairness as each node has equal opportunity to participate and redeem their individual reputations, which naturally levels the playing field more so than alternative methods.

A blockchain-based machine learning framework for edge services (BML-ES) proposed by Tian et al. [12] provides a solution for computationally weak IIoT devices to participate in computing and aggregating ML models to produce decision trees. MnemoSys tackles a similar core issue, although our nodes in the network are not assumed to be weak. BML-ES chooses to have training nodes engage in a smart contract which locks a deposit with a task publisher, who may ask trainers to make a decision tree. MnemoSys opts to leverage a reputation-based incentive system, rather than a monetary-based system. Their framework includes a Proof-of-Training-Quality to solve the problem of lazy or malicious task training nodes, a large challenge we also face in this paper. Tian et al. [12] mentions how accuracy of calculations may be verified using test sets as well as a 'verification formula.' This formula has a binary output determining if a model is verified or not by using accuracy of the model as the determining factor. The accuracy of the model is a weak form of establishing trust. A model may have low accuracy, yet the training node was honest in the process of constructing it, perhaps caused from the original data set. We instead choose to employ a reputable quorum of trainers, who must reach a consensus on the model, or other computation, creating a more trustworthy process.

Other valuable research which utilizes a similar Boltzmann machine includes: Cheng et al. [13] with their Reduced Boltzmann Machine (RBM) probabilistic modeling performance analysis. In this work, they explore using the RBM as an approach for unsupervised generative modeling of both quantum and classical data sets, and in the process demonstrate the RBM as a robust modeling technique.

Singh et al. [14], uses a deep Boltzmann machine based flow analyzer in their proposed deep-learning-based blockchain framework. This Boltzmann machine is able to detect anomalous behavior within the system, providing an important quality of dynamic responsiveness in the face of such behaviour. This flexibility and responsiveness is provided in MnemoSys as well using a Boltzmann machine, but rather than anomalous behaviour, it is used to predict the outcome of a quorum based on node reputations. This similarly provides MnemoSys a dynamic and responsive quality in the face of low quality or malicious nodes.

III. Preliminaries

A. Consensus

Distributed and decentralized blockchain networks, such as Bitcoin, require mechanics for the majority of nodes in the system to have a nearly identical perspective of the system's state. This includes the blockchain, and more specifically, what data is on the blockchain at a given moment. It is important for the nodes to have a common view, so the network as a whole can operate as a cohesive group. Otherwise, unexpected behaviour or consequences may arise. For example, these networks may allow you to query the state of the blockchain by connecting to any single node, but if the network has disagreements about the blockchain, your response back from the node may be different depending on who you happen to connect to. One node may show you have 10 assets recorded to belong to you, while another says you have 5. This concept of a common agreement amongst many members is called 'consensus'. The way in which the members achieve the agreement is their 'consensus algorithm.' The most commonly used consensus algorithms today are PoW (Proof of Work) and PoS (Proof of Stake).

B. BFT

When implementing our consensus algorithm, the most important consideration was the ability to tolerate dishonest nodes who may try to bring the system down or who deceive other nodes for personal gain. Consensus algorithms which can tolerate up to one-third of nodes being malicious are considered to be Byzantine Fault Tolerant (BFT), named after the Byzantine Generals problem posed by Leslie Lamport and his colleagues in 1982. Being BFT allows us to continue system operations while faced with considerable amounts of bad or ill actors.

C. Quorum

One such way to achieve consensus on the system's state and scale alongside an increasing amount of nodes is to use a quorum-based consensus protocol. Consider a node to be a member within a set. A quorum refers to a subset of members selected in order to come to a consensus which will be representative of the whole system. What the quorum decides, the rest of the members should follow. Generally, a quorum follows the BFT consensus standard within the group, asking two-thirds of members to agree, tolerating one-third to be dishonest or ill. A quorum may also be required to reach unanimous consensus in order for other members to accept a decision. In MnemoSys, we ask the quorum to completely agree on the system's state, or to choose a new quorum.

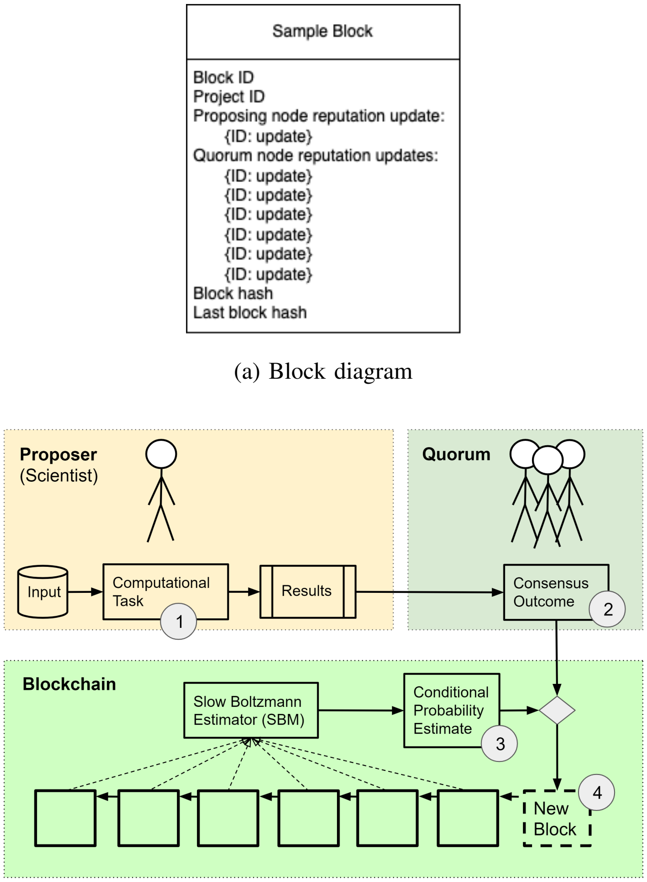
IV. Solution: MnemoSys

A. Solution Overview

There are three main components that form MnemoSys. First, proposer and quorum performance is recorded on a decentralized ledger; this is key to the formulation of the reputation score, as all recent successes and failures are considered. Reputation is then calculated using a weighted harmonic mean (WHM), with weights applied inversely to the expanding time windows. A quorum is then stochastically determined, and a confidence metric is produced using an abstracted application of a Restricted Boltzmann Machine (RBM).

B. Decentralized Ledger

In order to be fully trustless, this protocol leverages blockchain as an integral component for reputation management. Performance is recorded on a private permissioned chain irrespective to the outcome of the quorum. Figure 1 displays both the system architecture and the individual blocks being written to the ledger. Each block records the ID's of the block itself, participating nodes, and the project to which the computation belongs. For each of the participating nodes, their correspondent reputation adjustments are also written to the block.



(a) Block diagram

(b) System diagram

Figure 1: System and block diagrams displaying recording of relevant information for reputation management. Individual node ID's are recorded with commensurate reputation adjustments; by recording ID's of nodes and projects, both historical reputation and project history can be aggregated efficiently. The system diagram displays the four steps that result in the appending of the block to the ledger.

The system diagram in Figure 1 also displays the four-step mechanism that precedes the writing of the block to the ledger. First, a participant in the network (referred to as the "Proposer") submits a computation to be verified by the network. Six nodes are then stochastically selected from the network to serve as the verifying quorum. The quorum members will then independently perform the same computation submitted by the Proposer, arriving at the same or different result. Third, the SBE produces a conditional probability estimate based on the historical performance of the six nodes that compose the quorum. Fourth, a new block is appended to the ledger recording the aforementioned computational information and adjustments to the reputations of the proposer and quorum members, which depend on the results of steps two and three. The specific adjustments and outcomes are enumerated in the following section detailing the SBE.

Aggregating the reputation of a specific node or following the progress of a specific project is a trivial mechanism already used in many blockchain applications. For example, the Bitcoin ledger tracks transactions largely through UTXO's, leaving wallets to determine their balance by trawling the ledger and aggregating transactions moving to and from their specific address.

The main advantage of MnemoSys is it's focus on reputation as the underlying currency of the ecosystem in which the protocol operates. Adjustments are tracked on an immutable ledger; no input is taken from the participating nodes outside of their individual duties in the quorum or computation. This is an improvement on previous reputation models, where node operators are allowed to rate each other, leaving the door wide open for shilling attacks. In this protocol, the scores themselves are directly determined by both the quorum's results and the probability produced by the Boltzmann system.

C. Historical Reputation

Weighting historical performance is an open challenge approached in many different ways. Tit-for-tat and "strikes" policies oscillate between being too lax and too draconian; their rigidity allows some measure of gaming the system. An implementation must also be dynamically scalable to respond to volatility in network-wide performance. Moreover, recent node behavior must be weighted higher than less recent.

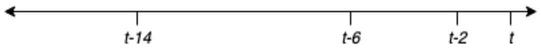


Figure 2: Time slicing of historical performance. This example uses $\epsilon = 2$ to produce windows of length 2, 4, 8, and so forth.

Our proposed solution uses geometrically expanding time windows moving into the past, with windows weighted inversely towards the present. Given some number of windows k and a scale factor ϵ, the weight of a specific window n is given as

$$w_n = \epsilon^{k-n} \qquad (1)$$

The aggregate sum of the weights is then produced as

$$\sum_{n=1}^{k} w_n = \sum_{n=1}^{k} \epsilon^{k-n} \qquad (2)$$

The average performance x_n over a window n is the arithmetic mean of the reputation scores within that time window, where the denominator is the length of the time window.

$$x_n = \frac{\sum_{t=\epsilon^{n-1}}^{\epsilon^n} x_t}{\epsilon^n - \epsilon^{n-1}} \qquad (3)$$

The inverse of the average normalized performance x_n^{-1} times the window weight w_n is then summed

$$\sum_{n=1}^{k} w_n x_n^{-1} = \sum_{n=1}^{k} \epsilon^{k-n} x_n^{-1} \qquad (4)$$

From this, the reputation θ of a specific node i is the WHM of the historical time windows, the weights of the windows w_n, and the average performance over those windows x_n

$$\theta_i = \frac{\sum_{n=1}^{k} w_n}{\sum_{n=1}^{k} w_n x_n^{-1}} \tag{5}$$

This system is inherently flexible and dynamic, allowing for manipulation of a key target variables – the scale factor ϵ. For example, when $\epsilon = 2$ and $k = 4$, the respective weights are $w_n = \{16,\ 8,\ 4,\ 2\}$. Setting $\epsilon = 3$ gives $w_n = \{81,\ 27,\ 9,\ 3\}$. Given an average historical performance of $x_n = \{0.5,\ 1,\ 1,\ 1\}$, $WHM_{\epsilon=2} = 0.6596$ while $WHM_{\epsilon=3} = 0.5990$. The number of time windows can be adjusted as well, though this has a less dramatic effect on the reputation calculation.

D. Slow Boltzmann Estimator

When a node proposes a computation to the network for validation, a quorum of six nodes is stochastically selected. The three highest reputations in this quorum are chosen as the "visible nodes," while the others are "hidden nodes." This labeling has less to do with their appearance and more with maintaining fidelity to the Boltzmann Machine nomenclature.

The RBM is a generative stochastic model that learns a probability distribution given a set of inputs. It is trained to maximize the expected log probability of a training sample selected from some training set. The Monte Carlo Markov Chain used to train an RBM begins at a random global configuration and selects units randomly, allowing them to stochastically update states based on energy gaps. It uses simulated annealing to seek out thermal equilibrium, where the probability of a global configuration is given by the Boltzmann distribution. The RBM does not use expectation-maximization like other commonly used systems, but seeks to minimize Kullback-Liebler divergence. Also called relative entropy, this is a type of statistical distance that measures the difference between a probability distribution P and a reference distribution Q

$$\begin{gathered} D_{KL}\left(P||Q\right) = H\left(P, Q\right) - H\left(P\right) = \cdots \\ \sum_{x\in X} p\left(x\right)\log\frac{1}{q\left(x\right)} - \sum_{x\in X} p\left(x\right)\log\frac{1}{p\left(x\right)} \end{gathered}$$

where $H\left(P, Q\right)$ is the cross-entropy of P and Q and $H\left(P\right)$ is the entropy of P. Minimizing KL divergence is equivalent to maximizing the log-likelihood, allowing the RBM to perform gradient ascent on the log-likelihood of the observed data.

In the Slow Boltzmann Estimator (SBE), nodes are stochastically determined and weights are functionally calculated. The entire quorum is treated as an adiabatic system, where energy is a direct result of the reputations of the consistent nodes. Given a visible node v_i and a hidden node h_j, the weight w_{ij} is the log of the average reputations θ

$$w_{ij} = \log_2\left(\frac{\theta_{v_i} + \theta_{w_j}}{2}\right) \tag{6}$$

Global system energy can then be calculated with the following formula

$$E = -\left(\sum_{i<j} s_i s_j w_{ij} + \sum_{i\in units} s_i \theta_i\right) \tag{7}$$

In the following probabilistic calculations, configurations of visible and hidden nodes are combinatorial expansions of all potential combinations of "good" (1) and "bad" (0) behaviors. For example, three visible nodes have 2^3 configurations $s_{i\in units} = \{111,\ 110,\ 100,\ 101,\ 011,\ 010,\ 001,\ 000\}$. Given visible node configuration α and hidden node configuration β, global system energy $E^{\alpha\beta}$ is calculated by

$$E^{\alpha\beta} = -\sum_{i\in units} s_i^{\alpha\beta}\theta_i - \sum_{i<j} s_i^{\alpha\beta} s_j^{\alpha\beta} w_{ij} \tag{8}$$

The resulting joint configuration probability $p\left(v^\alpha, h^\beta\right)$ over visible and hidden nodes depends on the energy of the joint configuration normalized by the partition function Z, which sums the energy of all possible configurations.

$$p\left(v^\alpha, h^\beta\right) = \frac{e^{-E^{\alpha\beta}}}{Z}, \quad Z = \sum_{\gamma\delta} e^{-E^{\gamma\delta}} \tag{9}$$

From this, we can derive the visible unit probability $p\left(v^\alpha\right)$ as the sum of the probabilities of all contained joint configurations

$$p\left(v^\alpha\right) = \frac{\sum_\beta e^{-E^{\alpha\beta}}}{Z} \tag{10}$$

In a system with three visible and three hidden nodes, the optimal visible node configuration $\alpha = 111$ has a maximal probability of 0.3907.

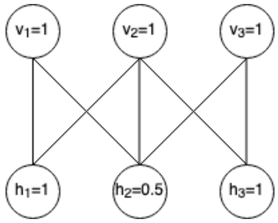


Figure 3: Example 6-node quorum with corresponding reputations

This leads to the most powerful dynamic parameter of MnemoSys; accepting or rejecting the results of the quorum is contingent on the visible unit probability staying above a decided threshold δ. For example, setting $\delta = 0.3$ will reject the results of all quorums that produce a probability below this. Reputation is therefore adjusted based on the outcome of both the above estimate and the following quorum consensus. Assuming the proposing node offers a calculation $f(x)$ that arrives at some answer D, all quorum members $N_{p=6} = \{N_1, \cdots, N_6\}$ must arrive at the same answer D by performing an identical calculation $f'_{N_p}(x)$. This leaves room for five potential outcomes.

Case #1: $p(v^{\alpha}) > \delta \wedge \forall p, f'_{N_p}(x) = f(x)$
The quorum reaches consensus, unanimously finding the same answer that the proposing node found, and also reaches a confidence estimate greater than threshold. The proposed computation is approved; a block is written to the ledger appending a 1 to the reputations of the proposing miner and quorum participants.

Case #2: $p(v^{\alpha}) > \delta \wedge \forall p, f'_{N_p}(x) \neq (x)$. The quorum reaches a consensus, unanimously finding an answer that is not the proposed value, and also reaches a confidence estimate greater than threshold. The proposed computation is rejected; a block is written to the ledger appending a 0.5 to the proposing miners' reputation and a 1 to the quorum participants' reputations.

Case #3: $p(v^{\alpha}) < \delta \wedge \forall p, f'_{N_p}(x) = f(x)$. The quorum reaches consensus, unanimously finding the same answer that the proposing node found, but does not reach a confidence estimate greater than threshold. The proposed computation is rejected; a block is written to the ledger appending a 1 to the reputations of the proposing miner and quorum participants. This case specifically allows for the rehabilitation of the quorum members' reputations that resulted in the computation being rejected. It is effectively a reward for perceived "good-faith" behavior.

Case #4: $p(v^{\alpha}) < \delta \wedge \forall p, f'_{N_p}(x) \neq (x)$. The quorum reaches a consensus, unanimously finding an answer that is not the proposed value, and does not reach a confidence estimate greater than threshold. The proposed computation is rejected; a block is written to the ledger appending a 0.5 to the reputation of the proposing miner and a 1 to the quorum members' reputations. Rather than simply re-selecting a quorum, this case specifically functions to penalize poorly performing quorums. Their low SBE estimate is a further indicator of previous sub-par performance.

Case #5: No unanimous consensus on $f'_{N_p}(x)$. In this case, the participants of the quorum do not reach a unanimous value. The proposed computation is rejected; a block is written to the ledger appending a 1 to the reputation of the proposing miner. Assuming all the miners responded but with different values, a 0.5 is appended to all their reputations. If a subset of the quorum is non-responsive entirely, a 0 is appended to their reputations and a 1 is appended to the other participants' reputations.

V. Experiments

A. Reputation Scoring

In order to determine whether the historical reputation system has the desired effects and modularity, controlled experiments were performed using randomly generated performances. To simulate both ideal and sub-optimal performances, four distributions were selected – pareto, power, poisson, and log-normal. Using these four historical performances, the WHM was calculated on a rolling basis.

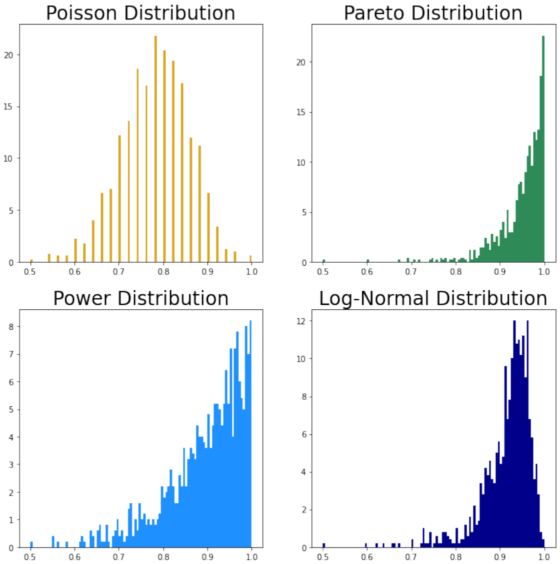


Figure 4: Four randomly generated distributions used to test historical reputation scoring.

B. Slow Boltzmann Estimator

In order to confirm the theorized functionality of the applied Boltzmann optimization process, a iterative testing approach is required. First, the size of the quorum must be determined and parameterized. The amount of visible nodes was varied between 3 and 8; the amount of hidden nodes was varied between 2 less than the visible nodes to 1 more than visible nodes; this resulted in 24 total configurations to be analyzed. From this, the optimal number of visible nodes was found to be 3 as detailed in the Results/Analysis section.

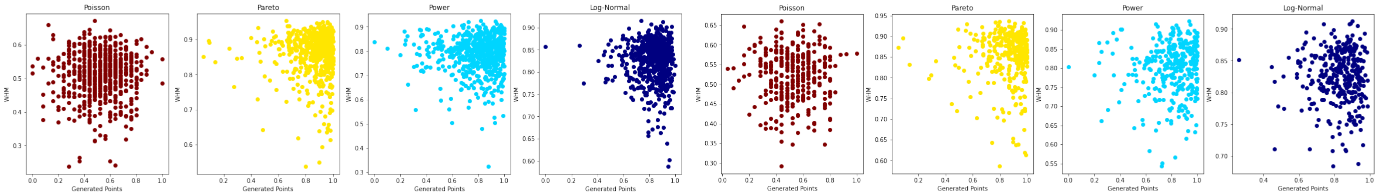
The next point to be determined was the number of hidden nodes, based on the probabilistic estimate resulting from the quorum calculations. Visible node reputations were clamped at a perfect 1.0, while hidden node reputations were set to 0.6.

C. Network Optimization

A similar approach was utilized towards quantifying the effect this protocol had on the network as a whole. Three different experimental modes were evaluated.

1) All communication between nodes was subject to a global "misfire rate" ϕ_g. For example, given $\phi_g = 0.05$, a polled node would return 1 with probability $1 - \phi_g$ and return 0 with probability ϕ_g
2) Randomly selected nodes were tagged as "malfunctioning," and these tagged nodes were given a specific misfire rate ϕ_t. This misfire rate only applied to these tagged nodes, while untagged nodes performed perfectly
3) All communiciaton between nodes was subject to a global misfire rate ϕ_g, and randomly selected nodes were tagged with a specific misfire rate ϕ_t. For untagged nodes, their effective rate $\phi_e = \phi_g$, while for tagged nodes $\phi_e = \phi_g + \phi_t$.

The iterative process for all trials was as follows.



(a) $\epsilon = 2$

(b) $\epsilon = 3$

Figure 5: Dynamic epsilon scaling's effect on reputation using four identical distributions

1) One node is chosen at random to be the "proposer"
2) Proposer node is polled, returning 0 with some probability ϕ_e and 1 with some probability $1 - \phi_e$
3) Six nodes are chosen at random to form the quorum
4) Joint conditional probability $p(v^\alpha)$ is produced resulting from the historical performances of the quorum members
5) Quorum nodes are polled, each returning 0 with some probability ϕ_e and 1 with some probability $1 - \phi_e$
6) Polled values are considered as agreement (1) or disagreement (0), and reputations for proposing and quorum nodes are adjusted as detailed in Solution - SBE section.

Two specific parameters were targeted in these trials. First, the clipping threshold represented the reputation level at which nodes were deactivated and removed from the network. Second was the SBE threshold below which quorums were not accepted, invariant of their consensus. These parameters were tested on their effect on global network mean reputation, number of active nodes, and quorums accepted.

As the protocol is permissioned by nature, Ethereum/Solidity was unfitting. Private permissioned options (Hyperledger Fabric, Gateway, etc) did not allow for adequate control over the consensus and quorum mechanisms. As a result, all experimentation was performed in Python 3.8 using a scratch implementation available on GitHub.

VI. Results/Analysis

A. Reputation Scoring

Since reputation is a lagging indicator ipso facto, there are certain effects we can predict will result from dynamically adjusting our scaling parameter epsilon. With a lower epsilon, we can expect reputation to move less in result to performance values further and further from an optimal score of 1. As seen in Figure 5, increasing the epsilon value increases the variability in the reputation values for the same performance histories. Because of it's lagging nature, measures of correlation are fairly useless - instead, we can qualitatively see the increased sparsity of the distribution.

By manually setting the historical performance values, we can cleanly see the effect of ϵ on reputation. In Figure 6, the perfect reputation history shows perfect reputation as expected. By adjusting the most recent time window's performance to an average of 0.5, we then see the resulting "near term misperformance" line. The convexity and decreasing value confirms the theorized effect that increasing epsilon reduces the reputation score. For example, setting $\epsilon = 2.0$ with the imperfect history gives a reputation score of 0.659; setting $\epsilon = 3.0$ reduces this to 0.599, a reduction of nearly 10%.

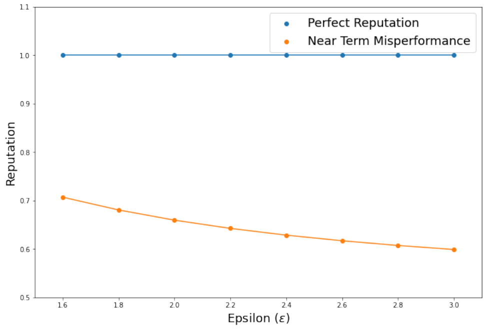


Figure 6: The effect of ϵ on reputation for perfect and imperfect history

B. Slow Boltzmann Estimator

The quorum size of six ("half jury") was selected expressly for mathematical efficiency. In this system, energy is calculated as the difference between state by reputation $(s_i \times \theta_i)$ and state by state by edge weight $(s_i \times s_j \times w_{ij})$. Increasing the number of nodes thus lessens the effect of one node's sub-optimal reputation on the probabilistic estimate. For example, quorum A with 3 visible and 3 hidden nodes has a maximal probabilistic calculation of 0.3907. Quorum B with 8 visible and 8 hidden nodes has a maximal probabilistic calculation of 0.0816. 3 visible nodes yield 2^3 potential configurations, while 8 visible nodes yield 2^8 potential configurations; all of these configurations have joint probabilities that are summed together in calculating the SBE estimate. A full sample calculation is shown in the Appendix.

Specifically increasing the number of hidden nodes yields a few different issues. First, it makes computing the normalizing term (partition function Z) less and less computationally feasible. Second, sub-par performance by hidden nodes drags down the computed score. For example, assume a quorum with perfect visible node reputations and hidden node reputations equal to 0.6. Quorum A with 3 visible and 3 hidden would have a probabilistic estimate of 0.2514; quorum B with 3 visible and 5 hidden yields 0.1702. In Figure 6, these two quorums are shown where "configuration 0" is the targeted joint probability - that all visible nodes are performing perfectly invariant of hidden node performance. Third, larger quorums increase probability of good-faith misbehavior, like latency issues or disconnection.

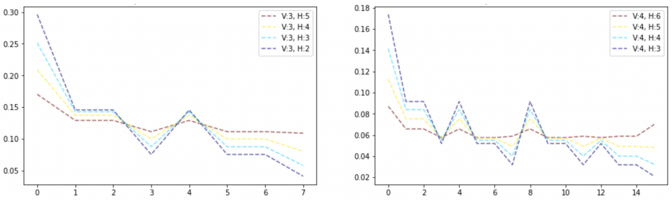


Figure 7: Various visible/hidden quantities (legend) and corresponding configuration joint probability (y-axis)

Once the visible node amount was set at 3, the number of hidden nodes was compared with the same experimental conditions as above. Visible node reputation was clamped at a perfect 1.0, while hidden node reputations were set to 0.6.

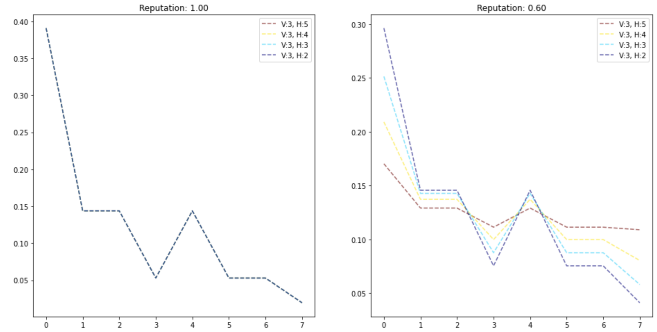


Figure 8: The effect of varying hidden node performance on corresponding configuration joint probability. Networks with more hidden than visible nodes had a more pronounced flattening of their probability curves for all configurations of correct and incorrect performance.

As seen in Figure 8, the reduction in hidden node performance lowered the calculated value of configuration 0 - a perfect visible node performance of '111'. One less hidden node maintained an identical maximal probability to perfect reputation performance, while two more hidden nodes had the worst performance. Clearly there is a balance to be struck between no exposure to node misperformance and over-exposure; an equal number of visible and hidden nodes provided this optimal balance.

C. Network Optimization

In training a Restricted Boltzmann Machine, the network seeks to minimize KL-divergence, which has the effect of maximizing log-likelihood. By inverting the usual operation of the RBM - stochastically selected nodes and computed weights, rather than constant nodes and stochastic weights - this protocol was theorized to maximize the targeted parameters of network-wide mean reputation. However, this could not come at the expense of draconian trimming of sub-optimal nodes, eventually rendering the network itself useless. Iterative progression in experimental parameters tested this hypothesis under increasingly realistic conditions, providing consistent results across all groups.

Figure 9 displays the results of the third mode of network optimization testing, where all communication was subject to a global misfire rate and randomly selected nodes were tagged with a specific misfire rate. The SBE threshold for quorum acceptance was 0.35, nodes were deactivated from the network below a 0.7 reputation, global misfire was 0.05, and tagged node misfire was 0.5. Networks were tested with 5%, 10%, 25%, 50%, and 75% of the nodes tagged as malicious. Figure 9(a) starts the entire network at a perfect reputation of 1.0, while (b) starts the network at 0.8.

When starting at a 1.0 reputation, we see an immediate dip in mean network reputation proportional to the percent tagged as malicious. Reputation then converges by 8000 iterations, aligning with a plateau in the number of active nodes. This indicates that the protocol was successful in removing the nodes tagged as malicious from the network. Table I shows the number of nodes remaining after 1000 iterations over each of the experimental conditions.

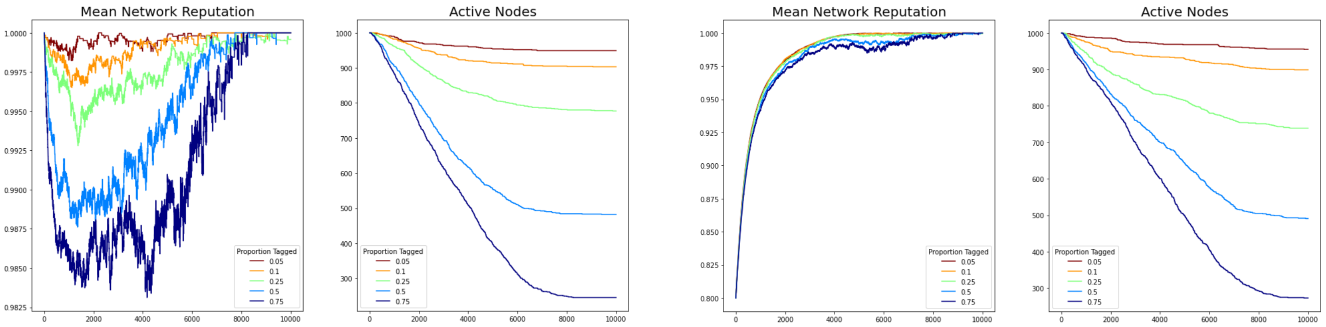
Tagged Portion	Active Nodes
0.05	949
0.10	903
0.25	777
0.50	482
0.75	245

Table I: Number of active nodes remaining after the 10000 iterations visually displayed in Figure 9. Note that the protocol was successful in removing all nodes tagged as malicious, except for 3 nodes in the 0.10 trial and 2 in the 0.25 trial.

While beginning the network at a 0.8 reputation is a relatively unrealistic condition, we can gain insight into the functionality of the protocol in the face of even more suboptimal network conditions. In the highest percentage condition, 75% of the network is malicious and malfunctioning 50% of the time. While it took closer to 8000 iterations instead of 6-7000 to remove all malicious nodes from the network, the protocol was still effective in rehabilitating mean network reputation with no non-malicious nodes removed.

VII. Conclusion

MnemoSys is a three-part contribution to the field of reputation systems, providing confidence for both the members and users of the network. Historical performance is first recorded on an immutable ledger tracking success, failure, reward, and punishment. Geometrically expanding time windows with exponential weighting preferentially consider near-term results. From this, a quorum of six nodes is abstracted as a restricted Boltzmann machine to produce a conditional probability estimate of good-faith behavior by the members. The results of both the estimate and quorum are then recorded back on the immutable ledger, closing the circle and moving on to the next quorum iteration.



(a) Starting reputation = 1.0

(b) Starting reputation = 0.8

Figure 9: Target network parameters after 10000 quorum iterations over 1000 nodes with SBE threshold at 0.35, reputation clipping threshold at 0.7, global network error rate at 0.05, and tagged node error rate at 0.5. In both starting conditions, the protocol converged, fully removing all malicious nodes and optimizing mean network reputation.

This protocol improves on previous reputation systems by taking the actual rating out of the hands of the nodes themselves, subverting potential malicious behavior like shilling attacks. Furthermore, the results derived from implementation of the protocol over an entire network are a testament to the mathematical power of probabilistic estimation. By abstracting a quorum as a closed system with computationally derived attributes, we can extrapolate a hard measure of confidence. Around this metric, we can scale parameters like historical window weighting, confidence acceptance threshold, and reputation clipping threshold.

Future work yields opportunities to build and improve on systems like this. For example, implementation over a fully distributed network with communication performed over TCP protocol would show how the gossip protocol actually allows the network to reach consensus on the ledger updates. Work can also continue on the mathematical formulation of the reputation and confidence estimate. Different formulas can be used for calculating weights between nodes in the quorum, rather than the logarithm of the average reputation. When tested over realistic conditions, a steady influx of new nodes can be introduced to the network with its own rate of malicious behavior.

By iterative application of the MnemoSys protocol, we can begin to solve some of the problems that face distributed networks. To ensure procedural integrity, an immutable ledger holds all historical performance; to protect against malicious behavior, under-performing nodes are removed from the network per their reputation score result; to provide robust network stability, parameters are able to be changed dynamically to respond to variable network conditions.

References

[1] M. Srivatsa, L. Xiong, and L. Liu, “Trustguard: Countering vulnerabilities in reputation management for decentralized overlay networks,” in *Proceedings of the 14th International Conference on World Wide Web*, ser. WWW ’05. New York, NY, USA: Association for Computing Machinery, 2005, p. 422–431. [Online]. Available: https://doi.org/10.1145/1060745.1060808

[2] R. Akbani, T. Korkmaz, and G. Raju, “Emltrust: An enhanced machine learning based reputation system for manets,” *Ad Hoc Networks*, vol. 10, no. 3, pp. 435–457, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570870511001867

[3] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, “The eigentrust algorithm for reputation management in p2p networks,” in *Proceedings of the 12th International Conference on World Wide Web*, ser. WWW ’03. New York, NY, USA: Association for Computing Machinery, 2003, p. 640–651. [Online]. Available: https://doi.org/10.1145/775152.775242

[4] A. Kaci and A. Rachedi, “Toward a machine learning and software defined network approaches to manage miners’ reputation in blockchain,” *Journal of Network and Systems Management*, vol. 28, pp. 478–501, 2020.

[5] A. Bugday, A. Ozsoy, S. M. Öztaner, and H. Sever, “Creating consensus group using online learning based reputation in blockchain networks,” *Pervasive and Mobile Computing*, vol. 59, p. 101056, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574119218306163

[6] T. Wang, J. Guo, S. Ai, and J. Cao, “Rbt: A distributed reputation system for blockchain-based peer-to-peer energy trading with fairness consideration,” *Applied Energy*, vol. 295, p. 117056, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306261921005134

[7] R. Dennis and G. Owen, “Rep on the block: A next generation reputation system based on the blockchain,” in *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2015, pp. 131–138.

[8] M. Li, H. Tang, and X. Wang, “Mitigating routing misbehavior using blockchain-based distributed reputation management system for iot networks,” in *2019 IEEE International Conference on Communications Workshops*

(ICC Workshops), 2019, pp. 1–6.

[9] Z. Zhou, M. Wang, C.-N. Yang, Z. Fu, X. Sun, and Q. J. Wu, "Blockchain-based decentralized reputation system in e-commerce environment," *Future Generation Computer Systems*, vol. 124, pp. 155–167, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X21001850

[10] J. Qi, F. Lin, Z. Chen, C. Tang, R. Jia, and M. Li, "High-quality model aggregation for blockchain-based federated learning via reputation-motivated task participation," *IEEE Internet of Things Journal*, pp. 1–1, 2022.

[11] Q. Zhang, Q. Ding, J. Zhu, and D. Li, "Blockchain empowered reliable federated learning by worker selection: A trustworthy reputation evaluation method," pp. 1–6, 2021.

[12] Y. Tian, T. Li, J. Xiong, M. Z. A. Bhuiyan, J. Ma, and C. Peng, "A blockchain-based machine learning framework for edge services in iiot," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 3, pp. 1918–1929, 2022.

[13] S. Cheng, J. Chen, and L. Wang, "Information perspective to probabilistic modeling: Boltzmann machines versus born machines," *Entropy*, vol. 20, no. 8, p. 583.

[14] M. Singh, G. S. Aujla, A. Singh, N. Kumar, and S. Garg, "Deep-learning-based blockchain framework for secure software-defined industrial networks," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 1, pp. 606–616, 2021.

VIII. Appendix

A. Sample SBE Calculation

	Visible Nodes			Hidden Nodes		
t	v_1	v_2	v_3	h_1	h_2	h_3
-2	1	1	1	1	0.5	1
...						
-6	1	1	1	0.5	1	1
...						
-14	1	1	1	1	1	1
...						
-30	1	1	1	1	1	1
...						
-62	1	1	1	1	1	1

Table II: Quorum node performance histories

			Visible		
			v_1	v_2	v_3
			1.0	1.0	1.0
Hidden	h_1	0.7948	-0.1561	-0.1561	-0.1561
	h_2	0.6597	-0.2692	-0.2692	-0.2692
	h_3	1.0	0	0	0

Table III: Corresponding edge weights for quorum nodes

Visible (α)	Hidden (β)	$-E$	e^{-E}	$p(v^{\alpha}, h^{\beta})$	$p(v^{\alpha})$
1 1 1	0 0 0	3	20.085	0.0192	0.3179
	0 0 1	4	54.598	0.0523	
	0 1 0	2.852	17.322	0.0166	
	0 1 1	3.852	47.089	0.0451	
	1 0 0	3.327	27.841	0.0267	
	1 0 1	4.327	75.680	0.0725	
	1 1 0	3.179	24.011	0.0230	
	1 1 1	4.179	65.270	0.0625	
1 1 0	0 0 0	2	7.389	0.0071	0.1468
	0 0 1	3	20.086	0.0192	
	0 1 0	2.121	8.341	0.0080	
	0 1 1	3.121	22.674	0.0217	
	1 0 0	2.483	11.973	0.0115	
	1 0 1	3.483	32.545	0.0312	
	1 1 0	2.604	13.515	0.0129	
	1 1 1	3.604	36.739	0.0352	
1 0 1	0 0 0	2	7.389	0.0071	0.1468
	0 0 1	3	20.086	0.0192	
	0 1 0	2.121	8.341	0.0080	
	0 1 1	3.121	22.674	0.0217	
	1 0 0	2.483	11.973	0.0115	
	1 0 1	3.483	32.545	0.0312	
	1 1 0	2.604	13.515	0.0129	
	1 1 1	3.604	36.739	0.0352	
0 1 1	0 0 0	2	7.389	0.0071	0.1468
	0 0 1	3	20.086	0.0192	
	0 1 0	2.121	8.341	0.0080	
	0 1 1	3.121	22.674	0.0217	
	1 0 0	2.483	11.973	0.0115	
	1 0 1	3.483	32.545	0.0312	
	1 1 0	2.604	13.515	0.0129	
	1 1 1	3.604	36.739	0.0352	
0 1 0	0 0 0	1	2.718	0.0026	0.0694
	0 0 1	2	7.389	0.0071	
	0 1 0	1.390	4.016	0.0038	
	0 1 1	2.390	10.918	0.0105	
	1 0 0	1.639	5.149	0.0049	
	1 0 1	2.639	13.996	0.0134	
	1 1 0	2.029	7.606	0.0073	
	1 1 1	3.029	20.679	0.0198	
1 0 0	0 0 0	1	2.718	0.0026	0.0694
	0 0 1	2	7.389	0.0071	
	0 1 0	1.390	4.016	0.0038	
	0 1 1	2.390	10.918	0.0105	
	1 0 0	1.639	5.149	0.0049	
	1 0 1	2.639	13.996	0.0134	
	1 1 0	2.029	7.606	0.0073	
	1 1 1	3.029	20.679	0.0198	
0 0 1	0 0 0	1	2.718	0.0026	0.0694
	0 0 1	2	7.389	0.0071	
	0 1 0	1.390	4.016	0.0038	
	0 1 1	2.390	10.918	0.0105	
	1 0 0	1.639	5.149	0.0049	
	1 0 1	2.639	13.996	0.0134	
	1 1 0	2.029	7.606	0.0073	
	1 1 1	3.029	20.679	0.0198	
0 0 0	0 0 0	0	1	0.0010	0.0336
	0 0 1	1	2.718	0.0026	
	0 1 0	0.066	1.934	0.0019	
	0 1 1	1.660	5.257	0.0050	
	1 0 0	0.795	2.214	0.0021	
	1 0 1	1.795	6.019	0.0058	
	1 1 0	1.454	4.282	0.0041	
	1 1 1	2.454	11.640	0.0111	

Table IV: Joint configuration probabilities