TontineCoin: Murder-Based Proof-of-Stake

1st Chris Pollett*, 2nd Thomas H. Austin*, 3rd Katerina Potika* and 4th Justin Rietz†
*Department of Computer Science
San José State University, San Jose, CA, United States
Email: chris@pollett.org, thomas.austin@sjsu.edu, katerina.potika@sjsu.edu
†Department of Economics, San Jose, CA, United States
Email: justin.rietz@sjsu.edu

***Abstract*—Proof-of-stake cryptocurrencies avoid many of the computational and environmental costs associated with proof-of-work protocols. However, they must address the *nothing-at-stake problem*, where a validator might attempt to sign off on competing blocks, with the hopes of earning coins regardless of which block becomes accepted as part of the blockchain. Cryptocurrencies such as Tendermint resolve this challenge by requiring validators to *bond* coins, which can be seized from a validator that is caught signing two competing blocks. Nevertheless, as the number of validators increases, it becomes more and more infeasible to effectively monitor all validators.**

In this work, we incentivize proper block monitoring by allowing validators to form *tontines*. Tontines are financial agreements where payouts to each member increase as the number of members decreases. In our system, a tontine is a group of validators that monitor each other's behavior, "murdering" any cheating tontine members to seize their stake. As the number of validators in a tontine is smaller than the number of validators in the currency as a whole, members can effectively police each other. We propose two methods whereby a Tendermint-like currency can be extended to allow for the creation of tontines: a pure proof-of-stake model, and a hybrid proof-of-stake/proof-of-work model. We describe snitch mechanisms for both the inter- and intra-tontine setting, argue our incentive mechanisms increase monitoring, and describe how it handles a variety of possible attacks.

I. Introduction

Bitcoin [16] has changed the way that we think about money. The Bitcoin protocol has provided a decentralized, resilient payment system. Its blockchain model has inspired new systems for handling sophisticated smart contracts [20], providing distributed storage [8], [14], [15], [18], and even such interesting applications as finding prime numbers [12] and building a public-key infrastructure [2].

However, Bitcoin's proof-of-work (PoW) based design has been the source of many problems. The irregular payment for the miners who maintain the blockchain has led to the formation of mining pools [17], reducing Bitcoin's decentralization. Its throughput is limited to one block of transactions for every ten minutes, causing scalability problems. The expense in terms of electricity usage has raised environmental concerns. Proof-of-stake (PoS) protocols seek to address these issues by instead tying the amount of coins a client has to its influence in the blockchain formation process. The idea is that the more coins a client possesses, the more vested that client is in the cryptocurrency's success.

While many variants of PoS protocols exist, Tendermint's approach is commonly adopted for other PoS systems. In this approach, validators *bond* coins for the right to produce and verify blocks of transactions. The validator is temporarily unable to spend the bonded coins, but receives shares in the rewards associated with the creation and validation of new blocks. If the validator misbehaves, other validators may write a transaction to the blockchain providing evidence of the cheating; the cheating validator is ejected from the set of validators and loses its bonded coins, which are divided up amongst the remaining validators.

Unfortunately, as the set of validators increases, so does the cost of monitoring other validators, making it increasingly likely that validators will elect not to spend the cost, and instead rely on the work of other validators.

To enforce fair behavior it is important to design a monitoring mechanism that detects cheaters and punishes them. Such a mechanism should make cheating less attractive. Moreover, the cooperation among users in order to detect cheaters must become profitable. The use of small groups [6], [9] is studied as a way to create an attack-resistant distributed system that is scalable, and might also improve robustness.

The blockchain technology, which is the basis of cryptocurrencies, has the potential of replacing various financial transactions, and if adopted more widely can have an impact on the lives of millions of people. So it is meaningful to investigate how old ideas of financial partnership, such as tontines, can be incorporated into the blockchain technology.

In this work, we describe TontineCoin, a new proof-of-stake protocol based on the financial structure of tontines. A tontine is an agreement where investors commit initial capital in exchange for a payout of funds over their lifetimes, with the unique property that the payouts are divided among the surviving beneficiaries. With this design, beneficiaries are (sometimes strongly) motivated to kill each other off, and thereby increase their payout.

While the economic incentives of tontines make them a questionable investment scheme, we find that it makes an excellent basis for a PoS protocol. Clients form tontines to earn the right to validate transactions, and hence gain an income. However, if a client in a tontine misbehaves, it can be kicked out of the tontine by the other members. Our design ensures that members of a tontine have a vested interest in detecting and announcing the bad behavior of other members, thereby addressing the nothing-at-stake problem. Furthermore, since the work required to monitor other validators is more limited, and the payout is divided amongst a smaller number of players, the expected benefit of monitoring for cheaters is not reduced as the number of validators increases.

This paper is organized as follows: Section II discusses the background to understand our coin design, especially Tendermint; Section III describes two models for tontines formation, the hybrid proof-of-stake/proof-of-work "train model" (Section III-A) and the pure proof-of-stake "mining cart model" (Section III-B); Section IV describes how tontines operate once formed; Section V describes the economics of our model; and Section VI concludes.

II. Tendermint Overview

In a PoS protocol, the creator of the next block is chosen either by a blockwise Byzantine agreement such as in Tendermint [13], the Ethereum Casper protocol [4], Hot-Stuff [1], Algorand [5] and StakeCube [6], or by a random chain-based consensus such as Ourobouros [11]. In this paper, we compare our design to Tendermint [3], [13], one of the most well-known PoS systems. We now provide an overview of the Tendermint design to facilitate understanding of our own design. We formulate our description of Tendermint to more closely match Bitcoin's widely used terminology rather than use exactly the same terminology used by Kwon [13].

Much of Tendermint's design follows that of other blockchains. Nodes participating in a Tendermint system are arranged in a network and relay new information to each other. Each node maintains a complete copy of a totally ordered sequence of events called a blockchain. Clients have accounts that hold some quantity of coins. Accounts are identified by a hash of the user's public key address. Clients may submit transactions to nodes to move coins from one account to another, typically offering some coins as a transaction fee. The number of coins held by an account can be determined from the blockchain by examining the number of coins transferred to and from an account.

Tendermint has four types of transactions: *standard transactions* are used to transfer coins to other clients; *bond transactions* are used by a client to offer coins as a surety bond, gaining the right to become a *validator* (roughly analogous to a miner in a PoW system); *unbonding transactions* are used by a client to regain spending access to bonded coins at the cost of losing the right to be a validator; finally, *evidence transactions* are used to announce a cheating validator.

Validators are responsible for making blocks and verifying the blocks produced by other validators in exchange for a reward. However, if the validator is caught cheating, it will lose all of the coins that it has bonded. Bonded coins cannot be used until a certain number of blocks have been created after the unbonding transaction to prevent a validator from quickly recovering its coins before its cheating can be detected. Evidence transactions contain two conflicting commit/vote signatures of a cheater or the cheater's signature on an invalid checkpoint. Once this transaction is accepted, the cheater's bonded coins are no longer spendable and the cheater is ejected from the set of validators.

The Tendermint blockchain is initialized with a *genesis block*, which identifies initial validators (defined below) and the quantities of coins that they have in escrow. Tendermint validators have voting power equivalent to the amount of coins that they have bonded. Validators in Tendermint are incentivized to include transactions in blocks by receiving a proportion of the transaction fees for each transaction according to the amount of coin they have in escrow. Besides using PoS rather than PoW, Tendermint also differs from Bitcoin in that it assumes users have accounts and that transactions transfer money between accounts rather than using Bitcoin's unspent transaction output (UTXOs) model.

A block in Tendermint consists of a header followed by a sequence of transactions. The header contains the block height (distance from the root block) and other meta data, the previous block header's hash, the root hash of a Merkle tree of validator signatures for the previous block, and a hash of the block's transactions.

Blocks are added to the blockchain using a variation of a Byzantine Agreement protocol given in Dwork et al. [7]. This protocol assumes a known upper bound Δ on times for messages to be delivered, and so a validator can use this in determining when to start processing a new block, when rounds end, etc. New blocks in the Tendermint variation of this agreement process are added in the following manner:

1) Each validator first determines who the other potential validators might be based on the current blockchain and who currently has bonded coins.
2) The validator pool is then reduced, by eliminating those potential validators whose minimum of the time since their bonding and the time since last participating in the validation process is greater than some parameter time Y.
3) Next each validator determines who the next proposer of a block should be. To do this an initial score for each validator is determined by the amount of coins bonded times the number of block proposal opportunities since bonding is computed. A final scores is computed by subtracting from this for each time that the validator was a proposer in the past the total value of bonded coins at the time they were a proposer. The validator with the largest final score is the next proposer and has a window of time in which to propose a block before the proposal calculation is redone.
4) To propose a block, a proposer sends the block together with their signature to all validators. At this point the Merkle root of signers is not yet filled in the block header. Validation proceeds in rounds until validators agree on a block of a proposer.

In each round, validators broadcast votes to all other validators consisting of their validator address, the block height, the round number, a proposed block, and their signature and then increments the round number. On receipt of these votes for a given round, an honest validator checks if they have previously fixed onto a proposed block. If so, their next round vote is this fixed onto this block. If not, they vote for the at most one valid block proposal received for that round. If the amount of coins in escrow of validators agreeing on the same block is more than $2/3$ of the total amount of coins in escrow, we say a $2/3$ majority is achieved. At which point, an honest validator should send a vote with a precommit flag and fix on to the block with the $2/3$ majority. If in a round an honest validator sees more than a $1/3$ vote weight for a different precommit block than their fixed onto value, the validator should unfix their block choice. If, on the other hand, they see a $2/3$ majority with precommit flags for the same block matching their own, then the validator should view the next block in the blockchain as having been determined and the sorted signatures of validators from the majority should be used to make the Merkle tree root for use in the next block's header. The honest validator should then increments its block height parameter.

Tendermint's PoS model eliminates some of the throughput issues of PoW models in that it does not depend on the somewhat hard to predict rate at which proofs of work are found and the related convergence issues this causes. Instead, throughput is largely determined by the message delivery rate Δ. On the other hand, in the Tendermint system one has to be careful about denial of service issues against proposers. There is also the issue that as the pool of validators gets larger the incentive to monitor the honesty of any particular validator goes down. The first of these problems is solved using a system of sentry nodes that act as intermediaries to prevent the direct address of the current proposer from being known [19]. In this paper, we are proposing using a tontine mechanism to solve the second problem.

III. TontineCoin – Tontine Formation

Clients in TontineCoin must form groups in order to join the set of validators, though once they have joined, they work independently from the other members of their tontine.

We outline two different approaches. In our *hybrid model*, clients form tontines off-chain and race to find a PoW. They then use this proof as a bid; the members of the tontine with the best bid join the set of validators once the bidding is complete. With the *pure PoS* model, clients instead submit a transaction to bond their coins and request to become validators. Once sufficient clients have requested to become validators, a new tontine is formed. The clients join the set of validators and remain in operation until the end of the tontine. For both models, the number of active tontines is bounded by m and we have a maximum duration of liveness of each tontine. While the intention is that these two models are alternative solutions, nothing prevents both approaches from being used in the same implementation if so desired.

A. Hybrid Tontine Cryptocurrencies – The Train Model

In our hybrid mode, we combine a PoW and PoS model. Clients band together to form tontines and collectively search for a PoW that will serve as a *bid* to become validators. Once accepted, the tontine members independently validate transactions in order to gain rewards. By integrating PoW into our system, we provide an additional form of Sybil resistance. Furthermore, we provide a way for clients without much stake to participate in the system and gain coins.

We refer to this design as the train model, since a new tontine is added to the validators every N blocks. In other words, the tontine "train" leaves at a regular schedule whether or not anyone is on board.

1) Hybrid Tontine Formation: In order to join the list of validators, clients must band together to form a valid tontine, where a tontine must have exactly S staked coins. A *bid* for a tontine includes:

- The *hash* of the block where the most recent tontine was selected.
- An *amount of coins* contributed by each tontine member, adding up to exactly S coins.
- The *share* of each tontine member.
- *Nonce*.

The block hash prevents a tontine from searching for a PoW for a block far into the future. Note that there is no target for the PoW. Instead, when the selection round is reached, the tontine with the best PoW is selected automatically.

The *share* of each tontine member is defined as the combination of the client id and the percentage of rewards that the tontine member will earn. A tontine needs both coins and computing power. Our protocol does not specify the trade-off in value between the two resources, leaving that to the tontine members to resolve. We expect that clients with more hashing power may require a larger share of the rewards relative to the amount of coins they have staked.

2) Hybrid Tontine Selection: When the tontine has found a PoW that beats the current best bid on the blockchain, they submit a transaction to the network; we call this transaction the *bid transaction*. As with other transactions, a transaction fee is typically required for this transaction to be accepted. This design also motivates the tontine to avoid spurious bid transactions.

The tontine with the current best bid for the next tontine selection is referred to as the *bid leader*. The process works as follows:

1) A client from the tontine submits a bid transaction, containing the bid information detailed earlier. The proof-of-work must beat the current bid leader's proof.
2) The block producer adds the bid transaction to the current block.
3) Once accepted to the block, the previous bid leader's coins are unbonded, potentially to be used in an additional bid transaction.
4) Once the selection block is reached, the bid leader is accepted, and its coins remain bonded until the end of the tontine's duration.
5) After D blocks delay, the new tontine begins operation,
6) As soon as the new tontine starts operation, the oldest tontine ceases to operate, and its coins are unbonded.

(Should there be no other active tontines, the remaining tontine may continue to operate indefinitely.)

B. Pure Proof-of-Stake Tontine Cryptocurrencies – The Mining Cart Model

In our second approach to tontine formation, we do not use a PoW to determine who gets to join a tontine. To form a tontine clients bond coins with a transaction as in the Tendermint approach. The two key differences with Tendermint are that the bonding transaction will have a tontine ID associated with it and that there is a maximum amount S that can be bound by a transaction. The tontine ID is an integer which must be one larger than the tontine ID of the last tontine, whose creation has been validated in the blockchain. A tontine with a fixed ID can begin validating transactions, that is, is started, when the number of coins bonded with its ID exceeds S coins. The first bonding transaction for a tontine ID that exceeds this quantity will be the last new member of this tontine. The proposer of a block gets to choose the ordering of bonding transactions within it, which determines which tontine a given bonding transaction belongs to. To facilitate identifying tontines, at the start of a block after the header, the block proposer lists the tontines formed in that block as well as the last bonding transaction hash of these tontines. Unlike the hybrid approach, the time at which a new tontine is created depends on when clients decide to bond coins, rather than at fixed intervals. We can imagine this as clients deciding to jump into a mining cart, and when the cart is full, starting to go along the rails. Hence, we call this version of tontine formation *the mining cart model*.

From our description above, it follows that in the mining cart approach a tontine can never have more than $2 \cdot S$ coins associated with it. It is also entirely possible for a tontine to be formed by a single client or for multiple tontines to be formed in a single block. In the case where all tontines have a single member, the mechanics of the PoS TontineCoin will largely reduce to that of Tendermint, where the minimum bonding transaction involves S coins. However, a client, with more than S coins to use in TontineCoin, who divides their coins amongst tontines with more than one member has an additional method of making money, catching cheaters, and hence is incentivized not to form single member tontines.

One remaining difference with Tendermint is that there are no explicit unbonding transactions in TontineCoin. Instead, coins are implicitly unbonded after T blocks have been validated or M newer tontines have been formed, whichever comes later. The first clause ensures that a member of a tontine gets to participate in the validation of a minimum number of blocks. The second clause in the implicit unbonding condition ensures that there are always a minimum of M tontines in operation.

See Figure 1 on how to form a tontine in the mining cart model. The first client is A with 8 coins, the second client is B with 10 coins and the last client is C with 4 coins. The tontine's total coins with ID 4548 is 22, which is bigger than S.

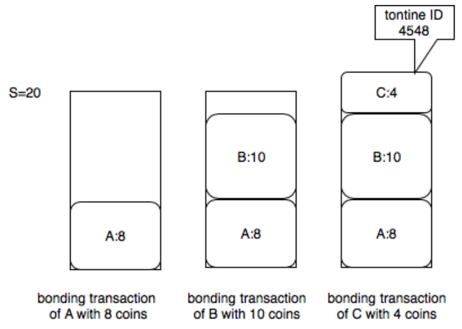
tontine ID 4548
S=20
A:8
B:10
C:4
bonding transaction of A with 8 coins
bonding transaction of B with 10 coins
bonding transaction of C with 4 coins

Fig. 1. A tontine with ID 4548, each client (rectangle), inside the rectangle a client ID with contributed coins, and $S = 20$.

IV. TontineCoin Operation

Once a tontine has been established, regardless of the approach used, the members validate transactions and monitor the behavior of other tontines. In this section, we review the operation of TontineCoin and the roles that tontines and their validators play.

A. Block Validation in TontineCoin

Like Tendermint and Bitcoin, both our TontineCoin variants have a genesis block. This block lists the initial set of tontines and their validator properties. Block validation in TontineCoin is done the same way for both our train and mining cart model. Block validation is like Tendermint in that a potential list of validators is determined to see whose turn it is to propose a block. Unlike Tendermint, the validator's score is with respect to their share within a tontine. So if a validator has money staked in two or more tontines, they will accordingly have two or more scores. When they become a proposer there is an associated tontine for the stake that allowed them to be the proposer. This fact will be later used when we describe our mechanism for handling cheaters. Other than this, the mechanics of block proposal and the voting to determine the next block is the same as the Tendermint model described earlier. Thus, block convergence of this protocol has the same guarantees as provided by Tendermint.

B. Rewards for Tontine Members

Validators are rewarded for their work with both transaction fees and with newly minted coins. We depart slightly from Tendermint's design, which only offers transaction fees. However, the number of newly minted coins created in a block is configurable, allowing us to match Tendermint by setting this value to zero.

Transaction fees in a block are divided evenly among all active tontines. Each member of a tontine is allocated funds according to their share within the tontine. However, the member only receives their reward if they sign off on the block. Otherwise, their share of the fees are burned.

C. Strategies for Rewarding Snitches

In Tendermint, validators who sign multiple blocks at the same height may be punished by having their stake seized. The seized coins may be divided up using different strategies, in the hopes of rewarding clients who provide evidence transactions. We refer to the clients who submit these transactions as *snitches*.

In a naive snitch-takes-all strategy, the snitch gains the cheater's staked coins. This has the benefit of strongly incentivizing the snitch to find cheaters, and also allows any client (and not just validators) to participate in monitoring the network. Unfortunately, a cheater can easily abuse this mechanism. By creating a second account, a cheater can "self-snitch" by forging a fraudulent transaction and then reporting it, thereby using the reporting mechanism to unbond without going through the usual unbonding process. Alternately, the cheater could wait until their cheating was identified, and once spotted, race the cheater to report the cheating to the network.

Instead, Tendermint divides up the rewards amongst all of the validators. This strategy avoids the problems with the snitch-takes-all approach. However, validators are only weakly incentivized to monitor each other, since the payout is minimal, and they receive the reward regardless. This approach also limits the pool of snitches to validators.

With the tontine strategy, the rewards are divided up amongst the tontine members, provided that the snitch is a member of the tontine. This approach more strongly encourages tontine members to monitor one another, but raises some risk of members colluding. We can address that concern by ensuring tontines are large enough to reduce the likelihood of colluding, and by having tontines monitor one another.

D. Inter-Tontine Monitoring

While tontines encourage greater monitoring within a tontine (intra-tontine monitoring), there is a risk of members of a tontine colluding to ignore each other's cheating, called the "joint cheating" strategy. Our first defense against this collusion is to revert to Tendermint's approach – any validator that is not a member of the tontine may still write an evidence transaction. In this case, the cheater loses their stake, which is divided amongst all validators in the network. However, while this approach ensures that the cheater always runs some risk of being detected, it lacks the additional incentive to monitor validators that our tontines were designed to introduce.

Our solution is to introduce inter-tontine monitoring, where tontines are randomly assigned to monitor the members of another tontine. We call the tontine that is assigned to observe another tontine the *watcher tontine*; the tontine being observed is the *monitored tontine*. Whenever a member of the watcher tontine writes an evidence transaction implicating the member of the monitored tontine, the cheater's lost stake is divided only amongst the members of the watcher tontine.

The assignment of tontines for monitoring is done as follows: periodically, tontines are ordered according to the hash of their tontine ID and the previous block hash[1]. Every tontine monitors the following tontine in the ordering, except for the last tontine, which monitors the first tontine.

[1]This ordering could potentially be biased by the block producer of the previous block. In future work, we plan to explore using randomized assignment, following approaches like Algorand [5].

We note that a similar approach could be used to have validators randomly assigned to monitor other validators without tontines. However, this approach would require more calculation on the part of validators to know whom they should be monitoring. Since individual validators are already monitored by the members of their tontines, we can keep the inter-tontine monitoring relatively infrequent, and thus less overhead for the validators.

E. Pre-evidence and Evidence Transactions

As with Tendermint, TontineCoin allows clients to write evidence transactions showing proof that a validator has attempted to cheat. However, new attacks may come into play since the rewards are not divided up amongst all participants.

We are particularly concerned that the block producer might attempt to steal credit for an evidence transaction from the client that originally identified the cheating. We refer to the client that first identified the cheating as the *snitch* in the rest of this section. Essentially, the block producer could throw away the snitch's evidence transaction, and use the details it provided to a new evidence transaction.

If the snitch and the block producer are in different tontines, it is in the economic interest of the block producer to steal the evidence transaction. The block producer might also have an incentive to steal the snitch's evidence transaction even if they are in the same tontine. If the block producer has accounts in multiple tontines, the block producer might gain a larger share of the seized tokens in a different tontine, in which case the block producer has an incentive to steal the evidence transaction and deny the snitch their reward.

In order to avoid these problems, we introduce *pre-evidence transactions*. The pre-evidence transaction contains a hash of the (still unposted) evidence transaction. After this transaction has been posted, the snitch then has a window of W blocks to submit the matching evidence transaction, gaining the reward after the W-block period has completed. Should two different clients post both pre-evidence and evidence transactions, the reward is given to the client that posted the first pre-evidence transaction.

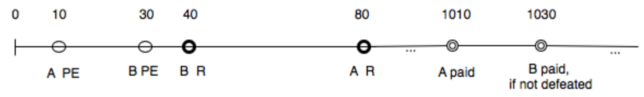

Fig. 2. Client A (the snitch) submits a pre-evidence (PE) transaction in block 10, and so does client B but later in block 30. The order of the evidence, denoted with R, is not important as long as A submits R before block 1010.

Figure 2 shows an example of this process where two users A and B both identify a cheater. In this example, we assume $W = 1000$ and that a validator cheats at block 0. Note that while user B reveals the cheater first, resulting in the cheater's tokens being seized at block 40, A will receive the reward at block 1010 since A's pre-evidence transaction was received first. However, if A failed to submit the evidence transaction,

B would receive the reward at block 1030. Our assumption is that B would not be able to suppress A's evidence transaction for the entirety of this period.

Note that A is not prohibited from posting an evidence transaction after block 1010, but loses priority. For instance, if A submitted their evidence transaction at block 1040 after B had already claimed the reward, then A receives no benefit.

V. Economic Analysis

After joining a tontine, per member expected payout is

$$E(payout) = q \cdot 0 + (1-q)\frac{ms(1+r)}{m(1-q)} = s(1+r) \qquad (1)$$

where q is the probability of being caught cheating, s is the cost to join a tontine, and r is the rate of return earned by the tontine (e.g. transaction fees). Thus, the expected payout is the same as an asset that pays a rate of return r.

However, the probability of death q is not independent of the tontine members' actions. Let e_i be the effort that member i puts into monitoring other tontine members, and let the probability of detection of cheating be

$$Pr(detect) = f(\Sigma_i e_i) = f(E) \qquad (2)$$

where f is a function such that $\frac{df}{dE} = f'(E) > 0$. That is, the probability of bad behavior being detected is strictly increasing in the effort members put into detecting such behavior. For member i, the probability of being caught is therefore $f(E_{-i})$, where $-i$ indicates all members but i, as we assume that an individual member does not monitor itself. Therefore, for member i, the expected payout is

$$E(payout) = (1 - f(E_{-i}))\frac{s(1+r)}{1-f(E)} - e_i \qquad (3)$$

However, if we assume that members are identical and don't monitors themselves, $f(E_{-i}) = f(E)$, leaving the expected payout of $s(1+r)$ minus the cost e_i. That is, members are better off if they can coordinate on not monitoring each other.

In order to avoid this "no monitoring" equilibrium, the cost to monitoring must be outweighed by the benefit from the perspective of an individual member. Starting from a position of no monitoring, member i will monitor if

$$\frac{s(1+r)f(e_i)}{1-f(e_i)} - e_i > 0 \qquad (4)$$

Thus, as long as a relatively small amount of effort can significantly increase the chances of catching a bad member and/or the payoff to being part of the tontine is high, members will have an incentive to monitor each other

Tontine members may coordinate on a "joint cheating" strategy when the payoff outweighs the net benefit even given the lack of multiple coordination channels (see Kandori [10]). In this case, inter-tontine monitoring may be effective. If the number of tontines is large and the assignment is randomly changed periodically, inter-tontine coordination becomes prohibitively difficult. A benefit of this mechanism is that it avoids the cost of a "one-to-all" relationship as with Tendermint.

VI. Conclusion

We have outlined how tontines may be used to improve monitoring within a PoS protocol. We have provided an overview of TontineCoin, a PoS protocol built around the tontine model, and shown two variants; a hybrid proof-of-stake/proof-of-work and a pure proof-of-stake model. Our results show that we can avoid a "no monitoring" equilibrium as long as the cost of monitoring is low compared to the expected return. TontineCoin helps push the incentives in this direction by both reducing the cost of monitoring by reducing the number of validators to monitor, and increasing the expected payout by dividing the seized coins among a smaller number of clients.

References

[1] Abraham, I., Gueta, G., and Malkhi, D. Hot-stuff the linear, optimal-resilience, one-message BFT devil. *CoRR abs/1803.05069* (2018).

[2] Ali, M., Nelson, J. C., Shea, R., and Freedman, M. J. Blockstack: A global naming and storage system secured by blockchains. In *USENIX Annual Technical Conference* (2016), USENIX Association, pp. 181–194.

[3] Buchman, E., Kwon, J., and Milosevic, Z. The latest gossip on BFT consensus. *CoRR abs/1807.04938* (2018).

[4] Buterin, V., and Griffith, V. Casper the friendly finality gadget. *CoRR abs/1710.09437* (2017).

[5] Chen, J., Gorbunov, S., Micali, S., and Vlachos, G. ALGORAND AGREEMENT: super fast and partition resilient byzantine agreement. *IACR Cryptology ePrint Archive 2018* (2018), 377.

[6] Durand, A., Anceaume, E., and Ludinard, R. Stakecube: Combining sharding and proof-of-stake to build fork-free secure permissionless distributed ledgers. In *Networked Systems - 7th International Conference, NETYS, Revised Selected Papers* (2019), pp. 148–165.

[7] Dwork, C., Lynch, N., and Stockmeyer, L. Consensus in the presence of partial synchrony. *J. ACM 35*, 2 (Apr. 1988), 288–323.

[8] Filecoin: A decentralized storage network. Tech. rep., Protocol Labs, August 2017.

[9] Jaiyeola, M. O., Patron, K., Saia, J., Young, M., and Zhou, Q. M. Tiny groups tackle byzantine adversaries. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2018), IEEE, pp. 1030–1039.

[10] Kandori, M. Social norms and community enforcement. *The Review of Economic Studies 59*, 1 (1992), 63–80.

[11] Kiayias, A., Russell, A., David, B., and Oliynykov, R. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Proceedings, Part I* (2017), pp. 357–388.

[12] King, S. Primecoin: Cryptocurrency with prime number proof-of-work. http://primecoin.org/static/primecoin-paper.pdf, 2013.

[13] Kwon, J. Tendermint: Consensus without mining, http://jaekwon.com/2014/05/11/tendermint/, 2014.

[14] Merrill, P., Austin, T. H., Thakker, J., Park, Y., and Rietz, J. Lock and load: A model for free blockchain transactions through token locking. In *IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)* (2019), IEEE.

[15] Miller, A., Juels, A., Shi, E., Parno, B., and Katz, J. Permacoin: Repurposing bitcoin work for data preservation. In *IEEE Symposium on Security and Privacy* (2014), IEEE Computer Society, pp. 475–490.

[16] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system, https://bitcoin.org/bitcoin.pdf, 2009.

[17] Rosenfeld, M. Analysis of bitcoin pooled mining reward systems. *Computing Research Repository (CoRR) abs/1112.4980* (2011).

[18] Storj: A decentralized cloud storage network framework. Tech. rep., Storj Labs Inc., October 2018.

[19] Tendermint documentation. https://tendermint.com/docs/tendermint-core/running-in-production.html#dos-exposure-and-mitigation, 2018.

[20] Wood, G. Ethereum: a secure decentralised generalised transaction ledger. https://gavwood.com/paper.pdf, 2014.