Peer-to-Peer Networking and Applications manuscript No.
(will be inserted by the editor)

TontineCoin: Survivor-based Proof-of-Stake

Chris Pollett · Thomas H. Austin · Katerina Potika⋆ · Justin Rietz · Prashant Pardeshi

the date of receipt and acceptance should be inserted later

Received: date / Accepted: date

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 increasingly infeasible to effectively monitor all validators, and to reach consensus.

In this work, we incentivize proper block monitoring by allowing validators to form *tontines*. In the real world, 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.

We extend our model to act as a validator delegated cryptocurrency, with the users having an incentive to partially participate. We show that these strategies

⋆ Corresponding author.

Chris Pollett, Thomas H. Austin, Katerina Potika, Prashant Pardeshi
Computer Science Department, San Jose State University
E-mail: chris@pollett.org, E-mail: thomas.austin@sjsu.edu,E-mail: katerina.potika@sjsu.edu,E-mail: prashant.pardeshi@sjsu.edu

Justin Rietz
Economics Department, San Jose State University
E-mail: justin.rietz@sjsu.edu

may benefit validators as well as speed up the block formation process. Moreover, we describe a prototype implementation of TontineCoin, and perform various experiments that support our theoretical analysis.

Keywords Proof-of-Stake · Cheater Detection · Cryptocurrency · Scalability

1 Introduction

Since its introduction, Bitcoin has changed the way that we think about money. By carefully aligning incentives, the Bitcoin protocol has provided a decentralized, resilient payment system. Its blockchain model has inspired new systems for handling sophisticated smart contracts [46], providing distributed storage [20, 32, 34, 43], and even such interesting applications as finding prime numbers [24] 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 [39], reducing Bitcoin's decentralization. Its throughput is limited to one block of transactions for every ten minutes, causing scalability problems. The increasing 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 Proof-of-Stake protocols exist, Tendermint's approach is commonly adopted for other Proof-of-Stake 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. Section 3.2 discusses Tendermint's design in depth.

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 problem of detecting cheaters is a problem that is studied in distributed multiplayer games [29, 47]. The use of small groups [14, 19, 21] is studied as a way to create an attack-resistant distributed system that is scalable, and might also improve robustness. As we describe in the related work, our system is novel with respect to prior work in that it focuses on when people have incentive to monitor each other rather than trying to make groups that must reach consensus before a global consensus between groups is reached. Since our approach does not rely on multiple tiers of agreement, we argue that it will make for a more easily implemented system.

In this work, we provide 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. Not surprisingly, tontines have served as a staple of murder mysteries such as Agatha Christie's 1957 novel *The 4:50 From Paddington* and as a plot device in TV episodes such as The Simpson's Season 7 episode, *Raging Abe Simpson...* and the Archer Season 2 episode, *The Double Deuce*.

While the economic incentives of tontines make them a questionable investment scheme, we find that it makes an excellent basis for a Proof-of-Stake 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 an expanded version of a conference paper [37]. We have added additional background information on tontines, described a prototype implementation of our TontineCoin, and proposed various validator strategies that define different levels of participation in the monitoring and in the block creation. A detailed analysis is presented on the payout users receive under the various strategies, and how that can lead to a form of validator delegation. Moreover, we perform experiments using one of these strategies, the *copy fast* strategy, to show that it is a viable validator strategy and to show the effect on the network's behavior as the number of *copy fast* validators increases; in short, we find that the network comes to consensus more quickly, but at the undesirable cost of reduced validation. Also, we add to the original paper a discussion of relaxed "murder" sentences for cheaters considering punishments like seizing parts of, rather than all of, the cheater's stake.

The organization of the paper follows: Section 2 reviews related work; Section 3 discusses the necessary background to understand our design, such as project SpartanGold (Section 3.1), Tendermint (Section 3.2) and tontines (Section 3.3); Section 4 describes two models for how tontines could be formed, including the hybrid Proof-of-Stake/Proof-of-Work "train model" (Section 4.1) and the pure Proof-of-Stake "mining cart model" (Section 4.2); Section 5 describes how tontines operate once formed; Section 6 gives an economic analysis of validator monitoring in our tontine setting, as well as considers validator strategies less than full monitoring and their incentive properties; Section 7 presents experiments for our proposed *copy fast* validator strategy and simulations of our TontineCoin approach; Section 8 contains implementation details of our TontineCoin; Section 9 extends the model to more forgiving "murder" modes; Section 10 discusses future directions for TontineCoin; and Section 11 concludes.

2 Related Work

Proof-of-Stake (PoS) protocols are rapidly being incorporated into real world distributed ledger systems, having been deployed in systems such as Cosmos with Tendermint [26], Ethereum with Casper [10] and Cardano with Ouroboros [23]. Other PoS protocols examples being considered for such systems include Chains-of-Activity [7], Hot-Stuff [1], Algorand[13], and StakeCube[14]. A summary of the characteristics of most of these protocols can be found in [36]. In this section, we consider some of these systems and how they relate to TontineCoin.

The Cosmos system [1] consists of a network of various independent blockchains, which are called zones. The zones use in their core Tendermint. Cosmos offers a light client option which has the same security as a full client, but with minimal resource requirements. A light client can receive proofs of the state of the blockchain from any full client. If a full node is disconnected it can use the Fastsync synchronization protocol, formally specified and model checked in [8], to learn blocks fast without participating in the consensus process. Both these approaches, i.e., the light clients and Fastsync, are different from our *copy fast* and *tontine monitor copy rest* strategies (described in Section 6.3) since nodes in our system are not predefined as full or light clients and so can switch from one strategy to the other in how they participate in the consensus.

The present paper performs some theoretical and experimental analysis of both the Tendermint and the TontineCoin system following in the tradition of earlier analyses of PoS protocols. The original Tendermint protocol is analyzed and modified by Amoussou-Guenou et al. [3] in order to implement one-shot and repeated consensus. Additionally, they reconsider the fairness of the protocol in the selection mechanism on the agreement set as well as the reward mechanism for the validators participating in the agreement. Ouroboros [23] proposes a PoS, which contains a provable security analysis. Moreover, it suggests a reward mechanism, such that the strategy of the faithful players to the protocol is an approximate Nash Equilibrium.

Though not strictly speaking a Proof-of-Stake system, Bitcoin-NG [16] attempts a similar approach to Tendermint in order to increase Bitcoin's transaction throughput. After a miner has announced a Proof-of-Work and produced a new block (dubbed a *key block*), it may produce additional *microblocks* without a Proof-of-Work until another miner finds a valid Proof-of-Work and takes control. As with Tendermint, if the miner produces conflicting microblocks at the same height, other miners may write an evidence transaction (called a *poison transaction* in their terminology).

Some other protocols, closer to our approach, aim to improve scalability by considering subsets or shards (groups) of users. In the former category belongs Algorand [13] that proposes a new Byzantine agreement protocol, which considers only a subset of users as participants for the consensus step. The participants are found by the use of verifiable random functions that select users in a private and non-interactive way, given the users have a private key. In the latter category belongs StakeCube [14] that combines users into shards, and chooses a constant size committee in each shard to perform the distributed steps of the consensus. In order to achieve an average sublinear communication per user, the number of

[1] https://cosmos.network

shards is chosen as $O(n^{1/2})$. We obtain a similar result on the number of tontines for the *tontine monitor ignore rest* strategy in order to keep the total number of messages linear (see Section 6.3).

Several approaches beyond PoS systems leverage groups of participants to make various systems resilient to attacks while still avoiding excessive communication and routing costs [12, 21]. In the distributed system considered by Jaiyeola [21] the size of each group is shrunk to $O(\log \log n)$, yet a fault tolerant system is created even under the assumption that a constant fraction of all the participants are faulty. By using PoW they manage to limit the amount of participants an adversary controls.

We conclude this section by mentioning that numerous technologies can be built on both PoS and PoW ledgers. The blockchain has been considered as an authorization database for a large-scale heterogeneous network [42], as a trusted service mechanism of a crowdsourcing system in 5G-smart cities [44] with smart contracts, and as a data sharing model for 5G drones [18].

3 Preliminaries

Cryptocurrencies can be classified in terms of how blocks of transactions are proven to be part of the blockchain of all transactions in the currency. Common strategies for this are Proof-of-Work, Proof-of-Stake, or some kind of hybrid between these. The integrity of the currency is determined by the incentives the verifiers of transactions in the currency, henceforth referred to as miners, have to behave honestly when determining which blocks constitute part of the blockchain, the number of verifiers, and the speed at which consensus is arrived at. In this section, we briefly survey well-known representatives of the Proof-of-Work and Proof-of-Stake approaches, discuss some of their strengths and weaknesses, and indicate why tontines may be useful to address some of the weaknesses. We also fix definitions and notations for the rest of the paper.

Bitcoin [35] is the most well known cryptocurrency and is an example of a Proof-of-Work system. In it, a transaction ledger is maintained as a nearly sequential tree of blocks. A block consists of two main parts: its header and a list of transactions.

The header consists of a version number, a SHA256 hash of a previous block in the Bitcoin blockchain[2], a SHA256 hash that is the root of a Merkle tree of SHA256 hashes [31] of transactions contained within the block, a time stamp for when the block was created, a difficulty target, and a nonce.

As we said earlier, after the header, a block contains a list of transactions. A transaction consists of a list of inputs and outputs each of which consist of bitcoin addresses together with auxiliary information. A bitcoin address is based on SHA256 hash of a public key. The inputs represent unspent transaction outputs (UTXOs) from earlier transactions. Each input contains a transaction ID referencing the transaction (and hence block) that contains the UTXO, its index within that transaction, a signature field proving the user can spend that UTXO, and a sequence number. This signature typically requires knowledge of the private key

[2] For the starting block, known as the *genesis block*, the previous block hash field has a special value.

corresponding to the public key used to make the bitcoin address. The outputs are new UTXOs that can be used in future transactions. They consist of an amount of bitcoin and the kind of signature needed to unlock these coins. This locking script information can be used to reconstruct a bitcoin address corresponding to this UTXO.

Each block is certified by a hashcash [5] Proof-of-Work. For Bitcoin, the hardness of this Proof-of-Work is specified by the difficulty target in the header. To satisfy the difficulty target the nonce must be set by the block miner to a value such that when the block header is hashed, the hash result has a number of leading 0's at least as long as required by the target. Although only the header is hashed, the integrity of the transactions within the block is ensured as the header contains the root of the Merkle tree of the transactions. The miner as per the version of the Bitcoin used by the block can specify as one of the transactions listed, a special transaction which "creates/mints" some new bitcoins, with an address that the miner could then use to spend in later transactions. Once a miner has created a valid block, the miner publishes it to the network of bitcoin miners. Honest miners, following the bitcoin protocol, when presented with chains whose headers all validate are supposed to deem the one with the larger total amount of work as the valid one. The process, by which consensus of which chain in the blockchain is the legitimate one, can be viewed as a Byzantine agreement process.

3.1 SpartanGold

SpartanGold [4] is a simplified blockchain-based cryptocurrency written in JavaScript. The goal of the project is to be a tool for rapid prototyping of new blockchain designs, as well as a tool for education.

SpartanGold's design is patterned after Bitcoin, though it is simplified to allow for easier experimentation. For instance, the blockchain only supports the moving of funds, and does not offer support for a scripting language.

The project is designed to run in either a single-threaded mode or run over a network connection. The former mode simplifies demonstrations and experimentation, while the latter version serves to provide a more realistic environment.

3.2 Tendermint Overview

Transaction throughput has been a major bottleneck in Proof-of-Work cryptocurrencies, like Bitcoin. The transactions per second is determined by the number of transactions that can fit into a block, the rate at which the difficulty targets increase given the size and processing power of the pool of miners, and the time for these miners to reach a consensus.

The Proof-of-Stake model attempts to address this bottleneck, while still maintaining the same decentralized nature of Proof-of-Work cryptocurrencies, like Bitcoin. Instead of Bitcoin's "one-CPU-one-vote" model, these protocols tie voting power to the amount of coins a client controls. For instance, Peercoin [25] relies on a variable Proof-of-Work, where a client with more *coin age* (the number of coins multiplied by the time since those coins were last spent) gets an easier Proof-

of-Work target. EOS [28] uses Delegated-Proof-of-Stake (DPoS) [27], where coin holders elect delegates responsible for maintaining the blockchain.

In this paper, we compare our design to Tendermint [9, 26], one of the most well-known Proof-of-Stake systems. We now provide an overview of the Tendermint design to facilitate better understanding of our own design.

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:

- *Send transactions* are used to transfer coins to other clients.
- With *bond transactions*, a client offers coins as a surety bond. It gains the right to become a *validator*[3] responsible for making blocks and verifying the blocks produced by other validators. However, if the validator is caught cheating, it loses all of the coins that is has bonded.
- The reverse of the bond transaction is the *unbonding transaction*; a validator regains its bonded coins and loses the right to be a validator. Bonded coins cannot be used until a certain number of blocks have been created after the unbonding transaction; this design prevents a validator from quickly recovering its coins before its cheating can be detected.
- If a validator is caught cheating, any client can post an *evidence transaction* showing the validator's two conflicting commit/vote signatures or its signature on an invalid checkpoint. A portion of the validator's coins are then destroyed and the validator is ejected from the set of validators.

The Tendermint blockchain is initialized with a *genesis block*, which identifies initial validators (defined below) and quantities of coins 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 Proof-of-Stake rather than Proof-of-Work, Tendermint also differs from Bitcoin in that it assumes users have accounts and transactions transfer money between accounts rather than in the Bitcoin setup, where UTXOs are used.

A block in Tendermint consists of a header followed by a sequence of transactions. We formulate our description of the header to more closely match our description of Bitcoin rather than use exactly the same terminology used by Kwon [26]. 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.

[3] A validator in a Proof-of-Stake system is roughly analogous to a miner in a Proof-of-Work system.

Blocks are added to the blockchain using a variation of a Byzantine Agreement protocol given in Dwork et al. [15]. 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.

Algorithm 1: Pseudocode for updating accumulated power

Precondition : accumPower: key-value pairs address & power
bondBalances: key-value pairs address, amount of coins bonded
proposerAddr: address of current block proposer
Postcondition: Block proposer's accumulated power decreased by total coins bonded
Other validators' power increased by amount of coins they have bonded.
Function : updatePower(accumPower, bondBalances, proposerAddr)

```
1 totalBonded := 0;
2 for (addr, amountBonded) in bondBalances do
3     accumPower[addr] += amountBonded;
4     totalBonded += amountBonded;
5 end
6 accumPower[proposerAddr] -= totalBonded;
```

When trying to come to consensus on a new block, each validator first determines who the other potential validators might be. This determination is based on the current blockchain and who currently has bonded coins. The validator pool is then reduced, 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.

The validators then go through multiple rounds in an attempt to arrive at a consensus for the next block. Each round consists of several phases: a proposal phase, where a single validator proposes a valid block; a prevote phase, where all validators attempt to lock on to the best block they see; and a precommit phase, where all validators attempt to commit to the locked block.

Proposal phase In the proposal phase, each validator determines who the next proposer of a block should be. For each validator, an initial score given by the amount of coins bonded times the number of block proposal opportunities since bonding is computed. From this result, a score is computed by subtracting for each time that validator was a proposer in the past the total value of bonded coins at the time they were a proposer. Algorithm 1 shows the pseudocode for this algorithm[4]. We update the block proposer's accumulated power by decreasing it and the rest of the validators' power by increasing it.

Prevote phase After waiting Δ time for block proposals, each validator must broadcast a prevote according to the following rules:

[4] We provide JavaScript implementations of this pseudocode and the pseudocode examples in the remainder of this paper for use with the SpartanGold framework. The implementation is available at `https://github.com/taustin/tendermint-sg`. The function names used in our pseudocode and in our JavaScript code are identical. Review of this implementation may facilitate better understanding of Tendermint's design.

- If the validator has locked on to a block from a previous round, they vote for that block.
- Otherwise, if they have received a valid block from the proposer, they vote for that block.
- Otherwise they prevote "NIL_VOTE" to indicate that no valid block was received.

Note that there should only be one valid proposal for a given block height and round. If conflicting proposals are received from the proposer, the validator may report this as Byzantine behavior.

Precommit phase After again waiting Δ time for prevotes, each validator must decide on the next step:

- If a block has gained 2/3 of the prevotes, the validator locks on to the block and broadcasts a precommit vote for that block.
- Otherwise, if "NIL_VOTE" has gained 2/3 of the prevotes, the validator releases any locks. No vote is broadcast.
- Otherwise, the validator does nothing.

Our pseudocode of this phase is shown in Algorithm 2.

Algorithm 2: Pseudocode for Precommit method

Precondition : prevotes: key-value pairs blockID (or NIL_VOTE) and amount of votes
roundNumber: number of attempts to reach consensus at this block height
delta: constant for waiting time multiplier based on latency
Postcondition: Validator is precommited to a block if there is a winning block.
Function : precommit(prevotes, roundNumber, delta)

```
1  winningBlockID := countVotes(prevotes);
2  if winningBlockID is undefined then
       // Failed to reach consensus; do nothing
3      else if winningBlockID == NIL_VOTE then
4          releaseLocks();
5      end
6      else
7          lock(winningBlockID);
8          broadcastPrecommitVote(winningBlockID);
9      end
10 end
11 wait (roundNumber * delta) msec ;
```

Commit decision At this point, the validator must decide whether to commit to the block, or to start a new round.

The validator again waits Δ time for precommits. If a block has gained 2/3 of the precommit votes, the validator broadcasts a commit vote for the block. At this point, the validator does not participate in additional rounds until the next block height; their commit vote counts as both their prevote and precommit votes for all subsequent rounds. They wait until they receive 2/3 of the commits from other validators before beginning a new block height.

If the validator has *not* received sufficient precommits, they begin a new round starting at the block proposal phase. The Δ time is increased for the next round, allowing the network to slow down when needed. The Δ time is reset for the next block height.

Tendermint's Proof-of-Stake model eliminates some of the throughput issues of Proof-of-Work 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 [45]. In this paper, we are proposing using a tontine mechanism to solve the second problem.

3.3 Tontine Overview

In this section, we briefly summarize the history of and concepts related to tontines. Much of the details here come from the excellent survey of McKeever [30]. A history of tontines starting in Holland and later Britain as well as comparison between tontines and other kinds of annuities can be found in Milevsky [33].

Several investment vehicles have gone under the name tontine. For example, in French influenced Africa and Asia, tontines usually refer to an Rotating Savings and Credit Association (ROSCA) in which members agree to contribute up to a fixed quantity M of money at regular meetings, and at those meetings, one member of the association receives all of that meeting's contributions. In one set-up, members are either live or dead. Live members bid a value less than or equal to M, and the lowest bidder receives the distribution for that meeting, but all live bidders only pay this low bid amount. Tie low bids are usually broken randomly. A member can receive this distribution only once, at which they become dead, and for all future meetings must pay M. When everyone who belongs to the tontine has received a payout, the tontine ends. This form of tontine dates to at least Tang dynasty (850s A.D.) China. It is alternatively formulated as the highest bidder winning the distribution for a given period, but where the remaining live members only pay $M - b$, where b was the winning bid. More on bidding equilibria for ROSCAs and this history can be found in Fang and Ke 2006 [17]. The modern tontine that we are interested in for this paper was proposed by, and gets its name from, Lorenzo de Tonti in the 1653. His idea, which he submitted to the Chief Minister of Louis XIV as a way to raise money for the French Government, was that subscribers would each buy an annuity at 300 livres per share and then nominate a third party as the life interest in the stake. These nominees would then be grouped by age ranges and would receive an equally divided annual payment based on the interest earned on the combined initial capital put up from members in that age range. As members in the given cohort die, the proportion of this annual payment for the surviving members would increase. Although de Tonti's proposal was never implemented, by the 1670s similar proposals had been established in Holland.

Tontines have been used to structure clubs, where the money contributed is used to purchase the facilities of the club, which are then used and owned by the members. They have been used as a form of insurance and estate planning. They have also been used for installment plan purchase scams. In the latter set-up, a cohort of individuals would make regular payments for a set time period. The members who never missed a payment for the whole period would receive either a diamond or 150% of their investment as their reward. As long as sufficiently many people miss a payment the selling company of this form of tontine could make money.

All of the tontines set-ups described above accrue benefits to the individuals that survive the longest, and incentivize these individuals to eliminate other tontine members. For this reason, tontines have acquired a bad reputation. In the early twentieth century in the United States, various states such as Wisconsin and New York enacted statutes that forbid deferred dividend insurance, where deferred in the former case meant a period of greater than five years, which implicitly targeted tontines [38]. Louisiana and South Carolina have actual statutes forbidding tontines. These statutes also provide a definition of a tontine as a policy that distribute benefits from a special fund to the oldest member of a division of its policyholders or the members of the division and class whose policy has been in force the longest period of time upon the death of a member in the division.

Tontines as used in TontineCoin defer benefits on a much shorter timescales than those mentioned above, on the order of hours or days, so likely are not governed by the former statutes. The benefits also do not accrue based on the death of a member, but instead on the provable discovery of cheating of a member, so likely do not run afoul of the latter statutes related to tontines.

4 TontineCoin – Tontine Formation

Unlike Tendermint, 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[5] 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.

[5] When comparing two Proof-of-Work values, we often describe it in terms of leading zeroes – the better proof produces more leading zeroes when the block is hashed. However, we actually compare the two hash values simply as numerical values, making the odds of a tie astronomically unlikely.

4.1 Hybrid Tontine Cryptocurrencies – The Train Model

In our first approach, the hybrid mode, we combine a Proof-of-Work (PoW) and Proof-of-Stake (PoS) model. Essentially, 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 Proof-of-Work 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.

This design bears a certain resemblance to mining pools – a collection of clients bands together to help earn rewards. However, the coordination only exists while the tontine competes for acceptance. Once the tontine is accepted, the clients independently validate transactions without coordination.

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.

4.1.1 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 ID of the block where the most recent tontine was selected. This field prevents a tontine from searching for a Proof-of-Work for a block far into the future.
- The amount of coins contributed by each member of the tontine, which must add up to exactly S coins.
- The *share* of each member of the tontine, defined as the combination of the client id, the computation power, and the number of coins stake they have contributed.
- A nonce, that when combined with the above fields produces a PoW.

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[6].

A tontine needs two distinct resources – coins and computing power. Our protocol does not specify the trade-off in value between the two resources, and instead leaves 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.

See an example in Figure 1 on how to form a tontine in the train model. Four clients with IDs A, B, C, and D and with different coins (respectively 4, 5, 8, and 3), and different hashing power (not shown) are bonding to form a tontine that has exactly S, where $S = 20$, total stake coins. After this group has produced the best PoW, they are accepted as tontine with an assigned ID. Note that after the tontine with ID 9742 is accepted, each of the members act as an independent validator.

[6] Our discussion assumes that only a single tontine is created in any given round. However, it is trivial to modify the protocol to accept the top X bids.

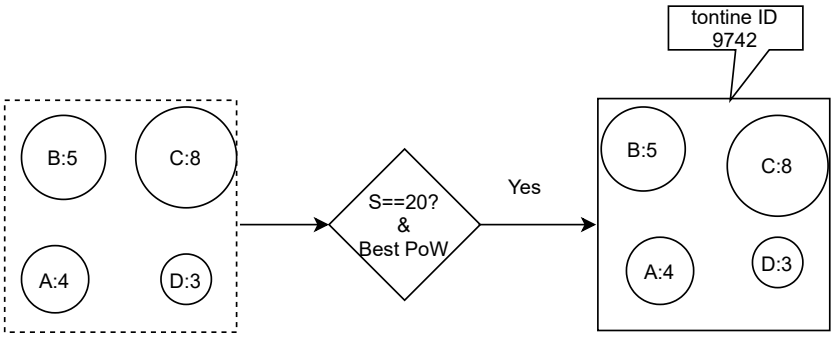

Fig. 1 Train model tontine formation example. Each client (circle), inside circle client ID: coins. And $S = 20$.

4.1.2 Hybrid Tontine Selection

When the tontine has found a Proof-of-Work 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[7].

The three first steps are for the bid leader selection, i.e., the tontine that is selected to begin operation. See the flow of these steps in Figure 2. A candidate tontine t competes to become the next tontine selected.

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

In our second approach to tontine formation, we do not use a Proof-of-Work 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

[7] Should there be no other active tontines, the remaining tontine may continue to operate indefinitely.

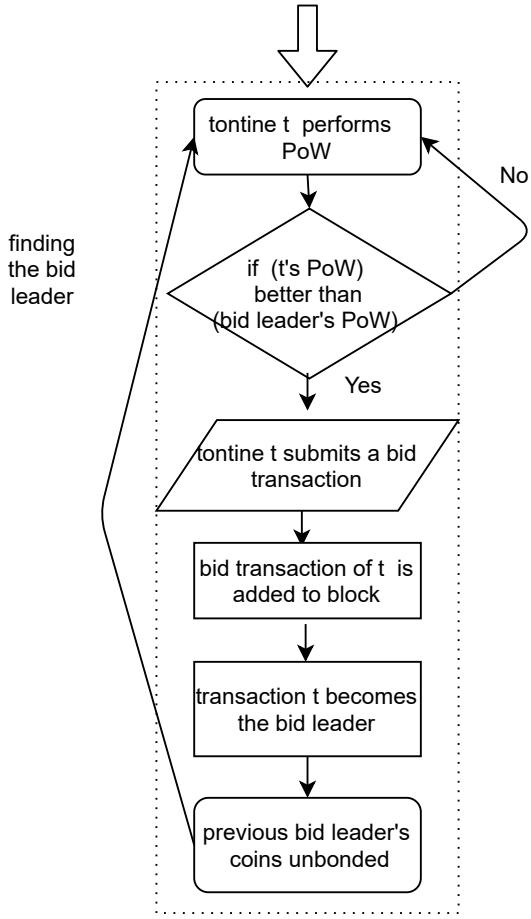

Fig. 2 Train model tontine selection. Flowchart of the bid leader selection.

– 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, it 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 view this design 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 in the mining cart model. 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 an example in Figure 3 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 total coins of the tontine with ID 4548 is 22, which is bigger than S.

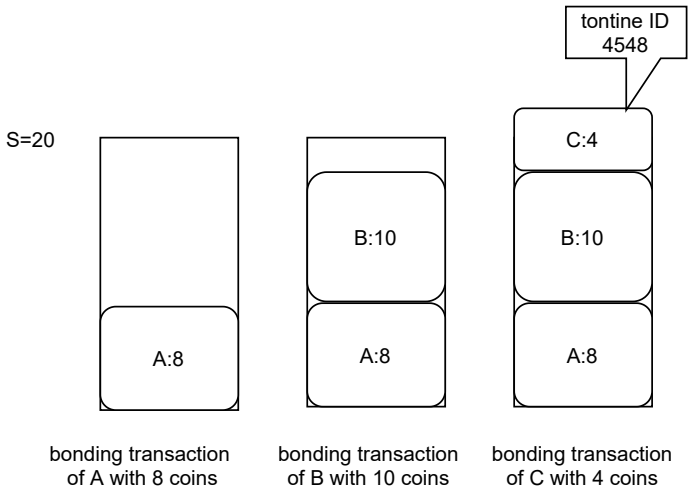

Fig. 3 Mining cart model formation example. A tontine with ID 4548, each client (rectangle), inside the rectangle client ID: contributed coins. And $S = 20$.

5 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.

5.1 Block Validation in TontineCoin

Like Tendermint and Bitcoin, both our PoW and PoS TontineCoin variants have a genesis block. This block lists the initial set of tontines and their validator properties such as how much stake each initial validator has. 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. Because of this, block convergence of this protocol has the same guarantees as provided by Tendermint.

5.2 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 tontine member only receives their reward if they sign off on the block. Otherwise, their share of the transaction fees are burned.

5.3 Strategies for Rewarding Snitches

In Tendermint, validators who sign multiple blocks at the same height may be punished by having their stake seized. In Kwon [26], this stake is then destroyed, however, 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 naïve 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 snitch to report the cheating to the network.

Another approach is to divide up the rewards amongst all of the validators. For the Tendermint protocol, 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 motivate one another, but raises some greater risk of tontine 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.

5.4 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. This risk is heightened by the lack of a form of Sybil resistance, since all members of the tontine might be controlled by a single person.

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, that 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 the hash of their tontine ID and the previous block hash. Every tontine monitors the following tontine in the ordering, except for the last tontine, which instead monitors the first tontine in the ordering.

5.5 Pre-evidence and Evidence Transactions

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

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[8]. In order to avoid this problem, 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.

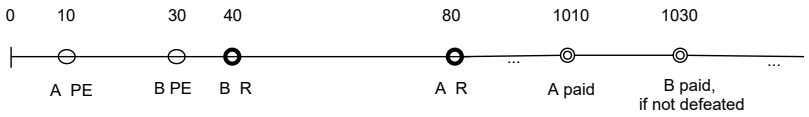

Fig. 4 An example on how the pre-evidence transactions work. Client A submits a pre-evidence (PE) transaction in block 10 and is therefore the snitch, and so does client B but later in block 30. Since, A submits the PE before B it will collect the reward after 100 blocks. The order of the evidence, denoted with R, is not important as long as A submits R before block 1010.

Figure 4 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, and A receives no benefit.

[8] 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.

6 Economic Analysis and Monitoring Strategies

Now that we have introduced our TontineCoin model, we'd like to analyze its incentive structures. We first begin with an analysis of the expected payout from participating in a tontine. This is used to further justify the inter-tontine monitoring of Subsection 5.4. We next discuss the potential benefits if a cryptocurrency makes uses of some kind of validator delegation. We then consider how plausible validator strategies in the TontineCoin setting naturally lead to the delegation of block production for any given block to a smaller subset of validators, while avoiding the potential collusion problems that might occur with a more direct delegation model.

6.1 Inter-tontine Monitoring Prevents No Monitoring Equillibria

The expected payout for a member of a tontine is a function of the probability q of being caught cheating, the number of members m, a member's stake s, and the appropriate rate of return r (e.g. transaction fees or some other mining reward).

After joining a tontine, the expected payout a member receives is the probability weighted sum of the payout when caught cheating and when not caught cheating:

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

The first term on the right hand side of the equation, $q \cdot 0$, represents the fact that a member receives no payout if they are caught cheating. The payout that a member receives when they are not caught cheating is the total of all members stakes ms times the gross rate of return $1+r$ divided by the number of members still in the tontine $m(1-q)$.

Thus, and somewhat trivially, the tontine is actuarially fair in that the expected payout is the same as investing in a financial instrument that pays a rate of return r [40].

However, in the use of the tontine described above, the probability of death q is not independent of the actions of the members of the tontine. Let e_i be the effort that member i puts into monitoring other members of the tontine. And let the probability of detection of bad behavior be

$$Pr(detect) = f(\Sigma_i e_i) = f(E) \tag{2}$$

where f is some 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

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

However, if we assume that all agents are identical, and no agent monitors itself, it can be assumed that $f(E_{-i}) = f(E)$, in which case we are left again with an

expected payout of $s(1+r)$ minus the cost of monitoring e_i. This suggests that all members are better off if they can coordinate on not monitoring each other.

In order to avoid this "no monitoring" equilibrium, the cost of 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

$$\frac{s(1+r)}{1-f'(E)e_i} - e_i > s(1+r) \tag{4}$$

where $f'(E)$ is the derivative of $f(E)$ and therefore the marginal benefit of additional monitoring.

$$\frac{s(1+r)f'(E)e_i}{1-f'(E)e_i} - e_i > 0 \tag{5}$$

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[9].

As previously mentioned, tontine members may coordinate on a "joint cheating" strategy when the payoff to this strategy outweighs the net benefit of intra-tontine monitoring. This is a particular possibility when tontine sizes are small, member identities are known, communication between members is possible, and/or there is a credible external enforcement mechanism (see Kandori [22] and Camera and Casari [11] for discussions of coordination given anonymity, memory, and community enforcement mechanisms). In this case, a secondary mechanism of between or inter-tontine monitoring may be effective. A member of a tontine is assigned to monitor the member(s) of another tontine, and as long as the number of tontines is large and the assignment is randomly changed periodically, inter-tontine coordination would become prohibitively difficult. A benefit of this monitoring mechanism is that it avoids the cost of a "one-to-all" relationship as with Tendermint, thus reducing the monitoring costs for any given member and therefore for the network as a whole.

6.2 Validator Delegation

Suppose Tendermint or tontine members delegate validation to a subset of members, this subset receives an additional benefit of increased fees from processing transactions but also an increased cost of processing transactions (e.g. bandwidth costs, computing costs, etc.). Other non-validators still incur the costs of monitoring the validators, but this cost is reduced as the number of validators is smaller as not all members are processing transactions. The non-validators, however, lose out on transaction fees, so a cryptocurrency would need a mechanism to incentivize delegation where possible.

While similar to the analysis above, the addition of increased transaction fees for the subset of members who are validators may reduce the incentive to cheat as validators now have more to lose – the future stream of higher transaction fees

[9] We have abstracted away from the fact that as the effort put into monitoring increases, there is a two fold impact: 1) for a given level of cheating, cheaters are more likely to be caught, but 2) increased monitoring will likely lead to less cheating given the increased likelihood of being caught.

– if they are caught misbehaving. That is, their opportunity cost of misbehaving is greater given the increased revenue from processing a larger number of transactions. Therefore, a potential benefit of delegation is that delegating members will need to expend less effort, and therefore incur lower costs, monitoring each other. This is similar in some ways to the Shapiro-Stiglitz shirking model [41] in which employers offer their workers higher wages in order to increase the cost to workers of "shirking" and therefore lowers the employers' ongoing monitoring costs.

6.3 Monitoring Strategies

We would now like to explore natural strategies that validators in TontineCoin might follow and how they result in a form of delegation in TontineCoin. As a byproduct, we will also be able to compare the cost of monitoring in Tendermint and TontineCoin.

To start let's consider one simple way a Tendermint validator can avoid monitoring: The validator in the Byzantine agreement protocol rather that actually count the votes from all the possible participants in a given round, instead waits for a vote from a particular participant that is trustworthy, or, maybe just fast, and then votes the same way as that person. Such a validator never votes twice in a round, and otherwise, maintains consistency, so won't be caught cheating. One can see a validator might even be incentivized to do this, as by doing this, blocks might be created faster, and so there is the potential to accrue more fees. If too many validators do this, then only a small number of validators might actually be checking transactions – probably the fastest ones – and these could have an incentive to collude. Section 7 offers some details of how this strategy could be implemented.

There are a couple of mechanisms in TontineCoin that reduce the severity of this problem. First, we note that TontineCoin has rules on how long a tontine exists, the longer of T new blocks or M new tontine formations. Call this length of time D. So the payoff per tontine to a validator is bound to D and so maybe slightly less tied to the block creation rate. The second mechanism is that if a validator catches a cheater within their own tontine or the other tontine that their tontine is monitoring, the reward is split only amongst members of the same tontine, so would be higher than if it is split amongst all validators or if the cheater's share was burned, this in turn is less than the reward in the TontineCoin setting if the cheater was caught in a monitored tontine.

To analyze this further, let's call the strategy of copying the first vote received, *copy fast*, and the strategy of doing full monitoring, *monitor*. We also assume that not all machines are equally fast at validating. If there is a total of T that has been staked, and there are n validators each with equal stake $T/n < S < T$, where S is the tontine size threshold in TontineCoin, a caught cheater loses T/n. To keep things simple let's assume $m \cdot T/n = S$, where m is some integer. In the Tendermint setting, this T/n is burned. So there is no incentive not to use copy fast. If the reward were split equally, each remaining validator in Tendermint would receive a reward of $\frac{T}{n(n-1)}$. If the average rate at which cheaters per unit time is discovered is r, then if the speed up from *copy fast* produces more $\frac{r \cdot T}{n(n-1)}$ additional transactions fees, it would be the preferred strategy. In the TontineCoin setting, the through rate reward is determined as a linear combination of the reward if the cheater

belonged to a monitored tontine and the reward when the cheater did not belong to a monitored tontine:

$$r \cdot \left(\frac{2 \cdot S'}{T} \cdot \frac{T}{n \cdot (m-1)} + \big(1 - \frac{2 \cdot S'}{T}\big) \frac{T}{n(n-1)} \right)$$

for some $S < S' < 2 \cdot S$ where S' depends on the exact size of the two tontines in question. As $m < n-1$, there would be more incentive to use the monitor strategy.

One could imagine that the most likely cheater is the current block proposer. Given this, the payout for catching a cheater when the proposer is not from the tontine a validator is monitoring could be low enough that the validator is better off using the copy strategy, but when the proposer is from a monitored tontine, the validator is better off monitoring. Call this strategy *tontine monitor copy rest*.

When following any of these three strategies, the validator needs to send messages to all members of the validator pool. In Tendermint, the validator pool consists of validators who have either bonded coins in any of the last Y rounds or who have bonded coins and who have participated in block production in the last Y rounds. The tontine monitor strategy suggests a modification to the validator pool so as to reduce the overall network bandwidth used in block production. First, when a cheater is caught by a validator that was not from a monitored tontine, the cheaters coins are distributed equally amongst all those with bonded coins. Second, modify the validator pool so that it consists of validators who have bonded coins within the last Y rounds, those who have participated in the last Y block productions, or those members of the same tontine as the proposer or of the monitoring tontine of the tontine of the proposer, that have participated in block production in one of the last Y times the given tontine had a member to propose a block. These modifications makes it clear that validators can skip participating in the monitoring for rounds where the proposer does not belong to a tontine they are monitoring and yet still participate when the proposer does belong to a tontine they are monitoring. Call the strategy of only monitoring when the proposer is from a tontine you are monitoring and otherwise not participate, *tontine monitor ignore rest*. Notice this strategy has the salutory effect that it might result in fewer messages being exchanged in a give round if many people follow it. This in turn could speed up the production of blocks.

If all validators followed *tontine monitor ignore rest*, one could ask how tontines should be sized, so as to maximize the number of validators monitoring any given transaction, yet minimize the number of messages that need to be passed for Byzantine agreement. In the case of Tendermint, every validator monitors every other validator ("all-to-all"). So, potentially, they must monitor all messages the total number of monitoring is in the worst case is $O(n^2)$, since each user has to monitor $(n-1)$ other users.

For each validator that chooses the strategy *tontine monitor ignore rest* we save on the message exchange. In the best case those are all the validators except the ones that cannot choose this strategy, which altogether are the validators of two tontines: the proposer's and the watcher to the proposer's one. For this discussion, we will denote by M, the total number of tontines. Let us set $S = (n)^{1/2}$ for all tontines. So we will have $M = (n)^{1/2}$ tontines. Within a tontine each person is monitored by $(n)^{1/2} - 1$ the rest of the tontine users, and also monitored by $(n)^{1/2}$ user of the watcher tontine, thus total number of messages passed is $O(n)$. More generally, we can set $S = n^{1/k}$ and $M = n^{(k-1)/k}$, where k is a constant, and still

keep the number of messages to $O(n)$. Note that for tontines of size less than $n^{1/2}$ we always get less than $O(n)$ many total messages.

Another extreme to be considered is the case where all tontines have just one member. In this case, we again have $O(n)$ total messages, but now every validator is only monitored by one other validator.

7 Experiments

In this section, we show our experimental analysis. We first review the *copy fast* validator strategy in order to test its effectiveness and to better understand its overall behavior on the network. We then use an agent-based model to estimate the number of "checks" - and therefore effort and network usage - that would need to occur between validators in order to detect cheating under both the Tendermint and TontineCoin frameworks.

7.1 Evaluating Effectiveness of the *copy fast* Validator Strategy

The *copy fast* validator strategy described in Section 6.3 seeks to increase profits for a validator by reducing the work required. In this section, we show how that strategy could be implemented and discuss the effect on network behavior as an increasing number of validators adopt this strategy. In short, we note that the network is able to come to consensus more quickly as the number of *copy fast* validators increases. Our results suggest that this strategy is a practical attack of concern to Proof-of-Stake cryptocurrencies that use a similar design to Tendermint.

In this version of the Validator class, the validator does not attempt to make its own decisions on voting, instead copying the first prevote, precommit, and commit vote that it sees for a round. The rules for locking or committing are still followed in accordance with the validator's vote.

Note that the *copy fast* validators still propose blocks normally. As a result, they must also collect commit votes normally. If they did not collect commit votes, they would not know when to move to a new height in the blockchain, potentially missing their turn to propose a block.

Our *copy fast* validator implementation tracks which rounds it has cast a prevote or a precommit to avoid duplicate votes. It also tracks when it has committed to a block at the current height.

We review the code for precommit votes to illustrate this design. In contrast to the `precommit` function for standard validators, previously shown in Algorithm 2, this function does nothing except set the timer to listen for the next subround:

```
precommit(prevotes, roundNumber, delta):
  wait (roundNumber * delta) msec
```

Instead most of the logic for a precommit is moved to the `collectPrecommit` function, normally responsible only for validating and collecting precommit votes. The modified `collectPrecommit` function is shown in Algorithm 3.

In our experiments, we simulated a network where messages would be delayed and potentially received out of order, but where they would always arrive. We then ran 16 validators in our SpartanGold Tendermint implementation until they had

Algorithm 3: Precommit method for a validator using the *copy fast* strategy

```
   Function: collectPrecommit(vote, height, roundNumber)
 1 if havePrecommitted() then
 2     return;
 3 end
   // The copyVote method broadcasts a vote agreeing with the copied vote, if
      the validator has not yet voted this round. Returns boolean indicating
      whether copying was successful.
 4 success := copyVote(vote);
 5 if success & vote.height == height & vote.roundNumber == roundNumber then
       // Follow copied vote's choices on locking/unlocking block.
 6     if vote.blockID == NIL_VOTE then
 7         releaseLocks();
 8     end
 9     else
10         lock(vote.blockID);
11     end
12 end
```

# copy fast validators	Average time (seconds)	Median time (seconds)	Low (seconds)	High (seconds)	Std. Dev.
0/16	168.27	143.5	93	431	78.95
1/16	159.42	138.5	95	499	77.18
2/16	142.85	124.0	91	293	55.44
3/16	134.88	104.5	91	310	60.22
4/16	139.35	119.0	91	393	60.97
5/16	129.12	127.0	95	203	26.59
6/16	135.38	129.5	95	238	31.80
7/16	122.04	114.5	91	211	26.03
8/16	106.92	103.5	87	165	15.01
9/16	120.35	113.5	87	187	22.62
10/16	109.62	102.5	85	200	25.07
11/16	120.62	102.0	82	392	62.02
12/16	106.31	98.0	84	199	24.68
13/16	108.04	98.0	85	211	30.06
14/16	109.00	96.0	87	212	30.58
15/16	104.69	103.5	83	125	11.79

Table 1 Effect on consensus time of *copy fast* validators

come to consensus on 10 blocks in the blockchain, selecting 0-15 of the validators to follow the *copy fast* strategy. (If all validators follow this strategy, the network will never come to consensus). For each number of *copy fast* validators, the experiment was repeated 26 times. All experiments were run on a Apple MacBook Pro running OSX v. 10.15.7 with a quad-core Intel Core i7 processor and with 16 gigabytes of memory.

Our preliminary results are shown in Table 1. They suggest that the network achieves consensus more quickly as the number of *copy fast* validators increases, as Figure 5 shows in a graphical form. However, the variance between sample runs leaves room for uncertainty.

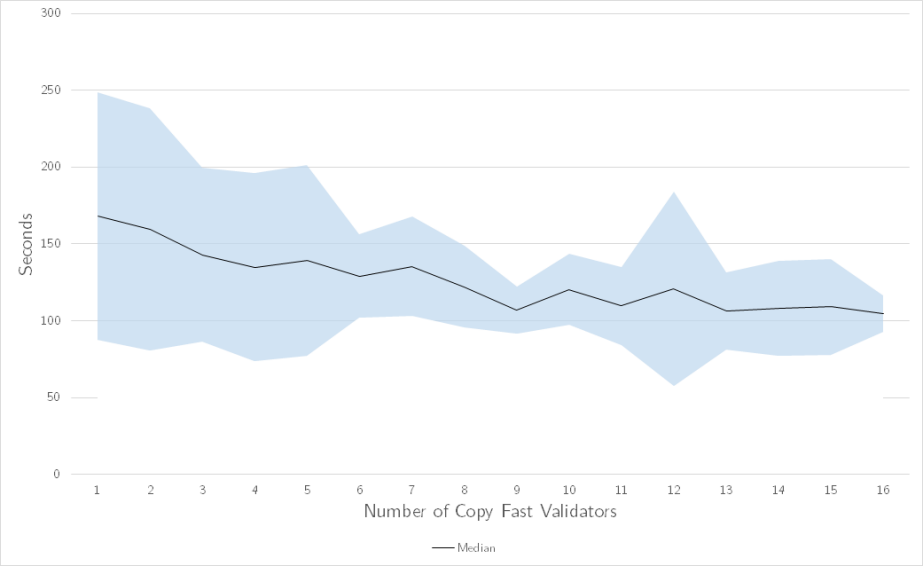

Fig. 5 Effect of *copy fast* validators on network consensus time

7.2 Simulations

One potential benefit of using small-group monitoring is that it reduces the effort needed in order to ensure that cheaters are detected. In order to estimate this benefit, we simulate cheating detection under Tendermint and TontineCoin using an agent-based model. Under both frameworks we assume 150 validators with each validator having an X% chance of cheating and a Y% chance that it is caught if "checked" by another validator. Moreover, the likelihood that a validator cheats is proportional to the number of remaining validators, i.e. we assume that as cheaters are caught, the remaining validators are less likely to cheat. In Tendermint, all validators check every other validator. However, under TontineCoin, validators only check all other validators within their tontine, and one validator per tontine is tasked with checking all members of another tontine. This setup is visualized in Figure 6.

We run 50 simulations for 10,000 time periods for each framework, well over the time needed for the number of cheaters detected to stabilize. That is, for TontineCoin we run the simulation until there is one member left in each Tontine as this provides the most conservative estimate of the average cost to catch a cheater. We set the percent change that a validator will cheat, X%, to 3% and the chance of being caught, Y%, to 50%. We find that on average Tendermint requires approximately 10,000 checks per cheater cost versus 6,000 under TontineCoin, suggesting that it is 66% more expensive to detect cheaters in Tendermint[10]. Using a standard t-test, this difference in means is significant at $p < .01$ (see Table 2).

[10] If we allow for the "murder" of cheaters under Tendermint, the cost per cheater is almost three times as expensive as checking occurs until there is only one validator left.

	Tendermint	TontineCoin
Mean	9,949	6,224
Median	9,948	6,298
St. Dev	76	739

Table 2 Number of checks per cheater caught.

8 TontineCoin Implementation

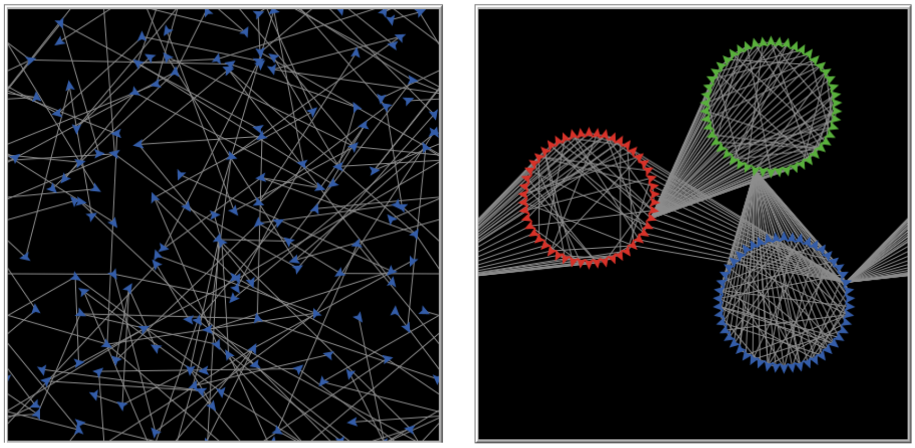

Fig. 6 Simulations: Tendermint (left) and TontineCoin (right).

We are currently working on developing a prototype of TontineCoin based on the SpartanGold framework, initially following the hybrid model described in Section 4.1.1. We describe our initial prototype here, noting this as ongoing work.

The TontineCoin fork of SpartanGold replaces SpartanGold's Proof-of-Work model with a Proof-of-Stake model akin to Tendermint's design. Unlike Tendermint, this implementation organizes validators into tontines as described previously. The implementation is available at `https://github.com/prashantp-git/TontineCoin`.

Transactions To support the Proof-of-Stake design, the TontineCoin implementation adds a `type` field. There is also a `data` field, whose contents vary by the type of transaction. Of these transaction types, all but `BID` transactions have direct equivalents in Tendermint's design.

The transaction types include:

- *REG* (for regular), indicating a transaction where coins are exchanged. This `data` field of this transaction includes the name of the receiver(s).
- *BID*, submitting a bid for a tontine of miners to join the validator set. The bid transaction includes the details of a newly formed tontine, detailed in Section 8.
- *BND*, for transactions where coins are bonded, and the client bonding the coins gains the right to join the set of validators, as well as the validator's share and

the ID of the tontine they belong to. The `data` field should contain bonding validator's name and public key.
- *UND*, for transactions where a client's coins are unbonded, and hence where the client is removed from the validator set.
- *EVD*, used for evidence transactions providing proof of a dishonest validator. The `data` field contains the cheating validator's name, the ID of that clients `BND` transaction, and the two conflicting blocks.

Unbonding (`UND`) transactions are used both for a client to unbond its own coins, and for the client's stake to be seized in the case where evidence of cheating has been provided. In the latter case, the ID of the evidence (`EVD`) transaction must be provided. (This design allows us to more easily model different approaches for seizing stake.)

Blocks Blocks in TontineCoin do not contain a PoW target or a coinbase transaction. Instead, the blocks have two additional fields – a set of previous block commit votes and the block creator's signature.

Validators The `Validator` class replaces the role of `Miner`. A validator maintains other validators' stakes, accumulated powers, and the total amount of staked coins, to be utilized in the consensus process. Additionally, they maintain:

- A ledger of other validators' coins to check if a validator posting a `BND` transaction has the specified amount of coins.
- Other validators' public keys to verify a block proposer's signature.
- Other validators' `BND` transactions utilized to retrieve a validator's `BND` transaction in case they post a `UND` transaction.
- A proposer and their proposal for the current round.
- prevotes and commits for the current proposal received from the other validators.
- commits for the previous proposal.
- The transactions received from clients to be added to the blockchain.

The above fields are equivalent to what is needed for Tendermint. In addition, a validator must track the ID of the tontine to which it belongs and its total share within that tontine. Finally, the validator must track the current bid leader in order to determine who the next tontine will be.

Tontine In our implementation, a tontine is made up of the following fields:

- `nonce`.
- `members`. This field specifies the name of the members along with their share in the tontine, the amount of coins staked, and their `BND` transactions.
- `id` of the tontine.

It is assumed that the validators' total stake is equal to a predefined amount given by `MAX_AMNT`, and that the total share is 1.

9 Forgiving Modes

In our discussion so far, we have proposed a fairly unforgiving model – the word "murder" implies a fairly severe and permanent punishment – but more relaxed approaches might be more appropriate in many situations.

Of course, high-quality service for the validators is very desirable. Setting more aggressive requirements can help to encourage this behavior. However, more aggressive requirements may mean more failure on the part of honest validators. For example, CosmosPool double-signed the same block [11]. As a result, CosmosPool was ejected from Cosmos's validator set and had 5% of its staked tokens seized. In the case of CosmosPool, the error happened because a back-up node and a primary node both proposed blocks at the same height.

The Tontine model can be adapted to a more forgiving design while still maintaining its benefits. Instead of seizing the entirety of a validator's stake on proof of misbehavior, we can configure the system to instead seize a portion of the validator's stake. Calibrating the punishment to be severe enough to be a strong detriment for malicious behavior, but not overly draconian for rare, honest mistakes is of course a balancing act, but one that blockchain engineers have been used to performing.

10 Future Work

While the fundamental goal of this paper was to increase the monitoring of validators, we see possible areas where the tontine structure might prove beneficial. We note that communication between nodes in a decentralized protocol can become a bottleneck. Limiting communication for inter-tontine coordination to a single node within each tontine could address that issue, but raise the risk that the current tontine node in charge of communication might be able to exploit its role. Potentially, the tontine murder-incentive structure could be used to keep these nodes honest.

Tontines might also be a useful structure for inter-communication between sidechains [6]. With this approach, a tontine could sit on two chains, serving as a currency exchange of sorts. If any tontine member attempted to abuse its power, the other tontine members could seize its staked coins on both blockchains.

Additional areas for future work involve improving the design of TontineCoin. For instance, it would be advantageous if TontineCoin members had identities to defend against Sybil attacks. Our bid process in the hybrid "train model" design works similarly to identity schemes used in protocols like Storj [43]. In future work, we will seek to introduce a similar mechanism to our protocol.

We also wish to develop a governance mechanism for TontineCoin. Tontines may serve a valuable role in this process as well; if we create a tontine of governing members, they could monitor one another's behavior and punish any abuse of their roles.

[11] `https://medium.com/@staked/slashing-risks-and-validator-diligence-f6901cc9622a`

11 Conclusion

In this paper, we have outlined how tontines may be used to improve monitoring within a Proof-of-Stake protocol. We have provided an overview of TontineCoin, a Proof-of-Stake protocol built around the tontine model, and shown two variants – a hybrid Proof-of-Stake/Proof-of-Work version and a pure Proof-of-Stake variant.

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.

Acknowledgements

We would like to thank Jae Kwon of Tendermint/Cosmos for his valuable feedback and insight, as well as the anonymous reviewers.

References

1. Abraham, I., Gueta, G., 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., Freedman, M.J.: Blockstack: A global naming and storage system secured by blockchains. In: USENIX Annual Technical Conference, pp. 181–194. USENIX Association (2016)
3. Amoussou-Guenou, Y., Pozzo, A.D., Potop-Butucaru, M., Tucci Piergiovanni, S.: Correctness and fairness of tendermint-core blockchains. IACR Cryptology ePrint Archive **2018**, 574 (2018)
4. Austin, T.H.: Spartangold: A blockchain for education, experimentation, and rapid prototyping. In: Silicon Valley Cybersecurity Conference (SVCC) (2020)
5. Back, A.: Hashcash - a denial of service counter-measure, http://www.hashcash.org/papers/hashcash.pdf. Tech. rep. (2002)
6. Back, A., Corallo, M., Dashjr, L., Friedenbach, M., Maxwell, G., Miller, A., Poelstra, A., Timón, J., Wuille, P.: Enabling blockchain innovations with pegged sidechains. `https://blockstream.com/sidechains.pdf` (2014)
7. Bentov, I., Gabizon, A., Mizrahi, A.: Cryptocurrencies without proof of work. In: International conference on financial cryptography and data security, pp. 142–157. Springer (2016)
8. Braithwaite, S., Buchman, E., Konnov, I., Milosevic, Z., Stoilkovska, I., Widder, J., Zamfir, A.: Formal specification and model checking of the tendermint blockchain synchronization protocol (short paper). In: 2nd Workshop on Formal Methods for Blockchains (FMBC 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2020)
9. Buchman, E., Kwon, J., Milosevic, Z.: The latest gossip on BFT consensus. CoRR **abs/1807.04938** (2018). URL `http://arxiv.org/abs/1807.04938`
10. Buterin, V., Griffith, V.: Casper the friendly finality gadget. CoRR **abs/1710.09437** (2017)
11. Camera, G., Casari, M.: Cooperation among strangers under the shadow of the future. American Economic Review **99**(3), 979–1005 (2009)
12. Castro, M., Druschel, P., Ganesh, A., Rowstron, A., Wallach, D.S.: Secure routing for structured peer-to-peer overlay networks. ACM SIGOPS Operating Systems Review **36**(SI), 299–314 (2002)
13. Chen, J., Gorbunov, S., Micali, S., Vlachos, G.: ALGORAND AGREEMENT: super fast and partition resilient byzantine agreement. IACR Cryptology ePrint Archive **2018**, 377 (2018)

14. Durand, A., Anceaume, E., 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, pp. 148–165 (2019)
15. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988). DOI 10.1145/42282.42283. URL http://doi.acm.org/10.1145/42282.42283
16. Eyal, I., Gencer, A.E., Sirer, E.G., van Renesse, R.: Bitcoin-ng: A scalable blockchain protocol. In: Symposium on Networked Systems Design and Implementation (NSDI), pp. 45–59. USENIX Association (2016). URL https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/eyal
17. Fang, H., Ke, R.: The insurance role of rosca in the presence of credit markets: Theory and evidence, https://www.ssc.wisc.edu/ scholz/seminar/rosca-wisc.pdf (2006)
18. Feng, C., Yu, K., Bashir, A.K., Al-Otaibi, Y.D., Lu, Y., Chen, S., Zhang, D.: Efficient and secure data sharing for 5G flying drones: a blockchain-enabled approach. IEEE Network **35**(1), 130–137 (2021)
19. Fiat, A., Saia, J., Young, M.: Making chord robust to byzantine attacks. In: European Symposium on Algorithms, pp. 803–814. Springer (2005)
20. Filecoin: A decentralized storage network. Tech. rep., Protocol Labs (2017)
21. Jaiyeola, M.O., Patron, K., Saia, J., Young, M., Zhou, Q.M.: Tiny groups tackle byzantine adversaries. In: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 1030–1039. IEEE (2018)
22. Kandori, M.: Social norms and community enforcement. The Review of Economic Studies **59**(1), 63–80 (1992)
23. Kiayias, A., Russell, A., David, B., 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, pp. 357–388 (2017)
24. King, S.: Primecoin: Cryptocurrency with prime number proof-of-work. http://primecoin.org/static/primecoin-paper.pdf (2013)
25. King, S., Nadal, S.: Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. http://primecoin.org/static/primecoin-paper.pdf (2012)
26. Kwon, J.: Tendermint: Consensus without mining, http://jaekwon.com/2014/05/11/tendermint/ (2014)
27. Larimer, D.: Delegated proof-of-stake (dpos) (2014)
28. Larimer, D.: Eos.io technical white paper. https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md (2017)
29. Laurens, P., Paige, R.F., Brooke, P.J., Chivers, H.: A novel approach to the detection of cheating in multiplayer online games. In: 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007), pp. 97–106. IEEE (2007)
30. Mckeever, K.: A short history of tontines. Fordham Journal of Corporate & Financial Law **15**(2), 491–521 (2009)
31. Merkle, R.C.: Protocols for public key cryptosystems. 1980 IEEE Symposium on Security and Privacy pp. 122–122 (1980)
32. Merrill, P., Austin, T.H., Thakker, J., Park, Y., Rietz, J.: Lock and load: A model for free blockchain transactions through token locking. In: IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON). IEEE (2019)
33. Milevsky, M.: King William's Tontine Why the Retirement Annuity of the Future Should Resemble Its Past (Cambridge Studies in Comparative Politics). Cambridge University Press (2015)
34. Miller, A., Juels, A., Shi, E., Parno, B., Katz, J.: Permacoin: Repurposing bitcoin work for data preservation. In: IEEE Symposium on Security and Privacy, pp. 475–490. IEEE Computer Society (2014)
35. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system, https://bitcoin.org/bitcoin.pdf (2009)
36. Nguyen, C.T., Hoang, D.T., Nguyen, D.N., Niyato, D., Nguyen, H.T., Dutkiewicz, E.: Proof-of-stake consensus mechanisms for future blockchain networks: fundamentals, applications and opportunities. IEEE Access **7**, 85727–85745 (2019)
37. Pollett, C., Austin, T.H., Potika, K., Rietz, J.: Tontinecoin: Murder-based proof-of-stake. In: J. Xu, S. Schulte, P. Ruppel, A. Küpper, D. Jadav (eds.) 2nd IEEE International Conference on Decentralized Applications and Infrastructures, DAPPS 2020, Oxford, UK, August 3-6, 2020, pp. 82–87. IEEE (2020)

38. Ransom, R.L., Sutch, R.: Tontine insurance and the armstrong investigation: A case of stifled innovation, 1868-1905. The Journal of Economic History **47**(2), 379–390 (1987). URL http://www.jstor.org/stable/2122236
39. Rosenfeld, M.: Analysis of bitcoin pooled mining reward systems. Computing Research Repository (CoRR) **abs/1112.4980** (2011). URL http://arxiv.org/abs/1112.4980
40. Sabin, M.J., Forman, J.B.: The analytics of a single-period tontine. Available at SSRN 2874160 (2016)
41. Shapiro, C., Stiglitz, J.E.: Equilibrium unemployment as a worker discipline device. The American Economic Review **74**(3), 433–444 (1984)
42. Shi, N., Tan, L., Li, W., Qi, X., Yu, K.: A blockchain-empowered AAA scheme in the large-scale HetNet. Digital Communications and Networks (2020)
43. Storj: A decentralized cloud storage network framework. Tech. rep., Storj Labs Inc. (2018)
44. Tan, L., Xiao, H., Yu, K., Aloqaily, M., Jararweh, Y.: A blockchain-empowered crowdsourcing system for 5G-enabled smart cities. Computer Standards & Interfaces **76**, 103517 (2021)
45. Tendermint documentation. https://tendermint.com/docs/tendermint-core/running-in-production.html#dos-exposure-and-mitigation (2018)
46. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger. https://gavwood.com/paper.pdf (2014)
47. Yeung, S., Lui, J.C., Liu, J., Yan, J.: Detecting cheaters for multiplayer games: theory, design and implementation. In: Proc IEEE CCNC, vol. 6, pp. 1178–1182 (2006)