TenderMint: Consensus without Mining

Jae Kwon
yk239@cornell.edu
Draft v. 0.3

Abstract. Cryptocurrencies such as Bitcoin enable users to submit payment transactions without going through a centralized trusted organization. The blockchain provides part of the solution, but much of the benefits are lost in securing the blockchain with computational proof-of-work mining which is needlessly expensive and slow. We propose a solution to the blockchain consensus problem that does not require mining by adapting an existing solution to the Byzantine Generals Problem.

1. Introduction

Cryptocurrencies have come into the spotlight since the introduction of Bitcoin [1]. The Bitcoin transaction log is secured by a network of miners who compete for rewards in the blockchain. This mining, or proof-of-work, comes with a hefty cost. At today's Bitcoin prices and reward schedule, miners are rewarded on the order of $1,500,000 a day to secure the blockchain – and a significant portion of that money is spent on electricity. Proof-of-work based consensus algorithms are also slow, requiring up to an hour to fully confirm a payment to prevent double-spending.

Other protocols (e.g. proof-of-stake protocols) have been proposed by the cryptocurrency community to solve this problem, but they typically suffer from the fallacy of false choices; nodes have nothing to lose by contributing to multiple blockchains, so consensus is not guaranteed. Unless there is actually "something at stake", all participants would be incentivized to sign any block that they encounter to earn fees. Yet other protocols suffer from assumptions of good behavior on the part of some participants, but these assumptions don't hold when the participants are financially motivated.

Our protocol overcomes the fallacy of false choices problem by requiring a surety bond deposit to participate in the consensus mechanism, ensuring consensus at every block, and strongly incentivizing participants to only sign the block agreed upon. We make a weak assumption about the participant's abilities to keep time, and we assume partial synchrony of the network. Our algorithm is based on a modified version of the solution to the Byzantine Generals Problem by Dwork, Lynch, and Stockmeyer [2], and is resilient up to $1/3$ of bonded coins belonging to byzantine participants.

2. Terms

Nodes are connected to each other in a peer-to-peer fashion and relay new information by *gossip*. Each node keeps a complete copy of a totally ordered sequence of events in the form of a *blockchain* as in Bitcoin. *Users (clients)* keep an *account* in the system, where the user's account is identified by the hash of the user's public key called an *address*. Each account can hold a sum of *coins* that can change with new transactions. Nodes relay new *transactions* as they are signed and submitted by users to a node of the network. There are 3 types of transactions.

- Send: Send some amount of coins from the signer's account to another.
- Bond: Lock coins as a surety bond.
- Unbond: Unlock the bonded coins.

A transaction is *valid* if it follows the rules of our protocol (e.g. sufficient funds to send, etc). Valid transactions are grouped into blocks. A block is valid if all the transactions in the block are valid. *Validators* are users with accounts that have bonded coins. We say that a validator has voting power in the amount of the bonded coins. Validators are *good* if the validator acts according to the protocol. Other validators are considered *byzantine*. Blocks are proposed and then *committed* into the blockchain by validators using the consensus algorithm. The network is *responsive* if transactions that pay sufficient fees get committed in a timely manner.

3. Validators

An account becomes a validator by posting some amount of coins as a surety bond. Once the bonding transaction is committed, the validator can participate in the consensus protocol with *voting power* in proportion to the amount of coins bonded. Bonded coins cannot be used in any transaction except for an unbonding transaction, afterwards the coins remain locked in the *unbonding period* of X blocks. If the validator fails to meet its obligations before the unbonding period is over, the validator can lose all of its bonded coins. The validator fails to meet its obligations if any of the following occur:

- Signing two conflicting messages at the same block height
- Signing an invalid checkpoint

Given the punitive nature of the algorithm and the long unbonding period, validators with significant voting power are unlikely to sign conflicting or invalid messages (at least until the unbonding period is over).

4. Consensus

4.1 On Byzantine Consensus

While most existing literature on byzantine consensus systems assume that each process is a discrete unit with equal weight and import, we extrapolate these studies into our problem domain where abstract processes (validators) have fractional presence in the form of voting power.

Fischer et al have shown in a seminal paper [3] that in an asynchronous system (where no assumptions are made about time) of deterministic processes, no protocol can guarantee consensus even with one faulty process. This is called the FLP impossibility result. Much research has gone into understanding ways to circumvent the FLP impossibility result by slightly modifying the problem domain, e.g. by sacrificing determinism, adding time, adding oracles etc [4]. Bitcoin circumvents the FLP impossibility result by making some assumptions about the synchrony of the network (i.e. nodes soon sync up with the network) and time (i.e. miners dedicate limited time and resources to the best blockchain). For example, if the Bitcoin network were such that the time for a block to be broadcasted takes longer than some multiple of the average block generation time, a minority mining pool with a superior connectivity can keep the network forked indefinitely.

Our algorithm is based on algorithm 2' from section 4 of [2] (Dwork et al). It assumes that the network is partially synchronous; there is assumed to be some unknown upper bound Δ on the time of messages to be delivered. We also assume that all non-byzantine nodes have access to an internal clock that can stay sufficiently accurate for a short duration of time until consensus on the next block is achieved. The clocks do not need to agree on a global time. It is possible to construct a consensus protocol with weaker assumptions about the validator's clocks [2], but we omit this possibility for simplicity. Like the algorithm as proposed by Dwork et al, it can tolerant of up to $1/3$ byzantine voting power.

4.2 On Byzantine Consensus

The blockchain is composed of sequential blocks connected by the hash of each block, which is computed by hashing the contents of the block.

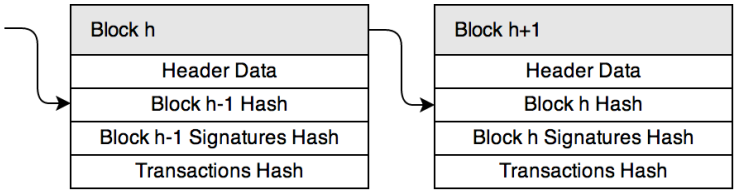

Figure 1: Block chain

A block is composed of a header (which includes information such as the block's height) and three hashes:

- The previous block's hash
- The root hash of a merkle tree of validator signatures for the previous block

- The hash of a list of new transactions

When a validator signs a block at height h, the signatures get hashed into a merkle tree and the result gets included in the next block. The signatures are ordered by the ordinal of the validator (i.e. by the chronological order of the validator's address), and missing signatures are denoted by an empty sequence of zeros.

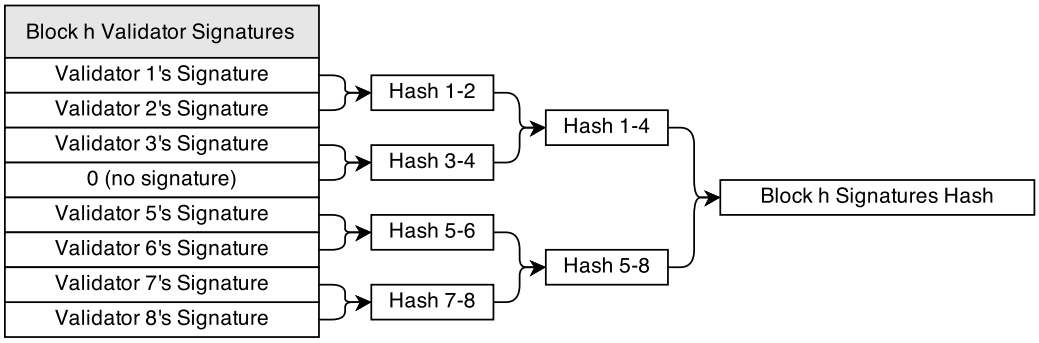

Figure 2: Signatures merkle tree

The transactions hash need not be a merkle tree.

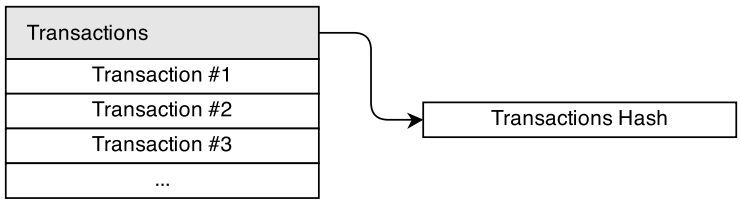

Figure 3: Transactions

4.3 Agreeing on the next block

After each validator sees that a more than 2/3 of the voting power has signed for the block(s) at height h-1, the consensus mechanism begins for the next block at height h.

Lemma 1: The consensus mechanism for block height h for all good validators begins within Δ of each other in global time.

The proof follows from the definition of Δ. The first good validator to see at least 2/3 of voting power for the previous block will broadcast all those signatures by gossip, thus all remaining good validators will see at least 2/3 of voting power for the previous block within Δ .

Let T be some fixed duration of time that is suspected to be at least 2Δ. We don't know what Δ actually is, so T is merely a guess that is baked into the algorithm. Let $W \geq T$ be some lower threshold on the amount of time between each successive block. The consensus mechanism begins by first waiting W, then proceeds in rounds until consensus is reached.

```
wait W & gossip votes for the previous block
loop starting with i=0:
  run consensus round i
```

At each consensus round, a single validator is chosen to propose a block. Any deterministic algorithm based on the existing blockchain history can be used to compute a total ordering of validators, but we want one that gives more priority to those with more voting power such that validators with less voting power have less chances to disrupt the responsiveness of the system. Before the consensus mechanism starts for the next block, a number and order is computed for each validator:

```
// calculate sorted list of validators:
sortedValidators = nil
for each block:
  for each validator:
    validator.accum_power += validator.power
    if sortedValidators != nil:
      sortedValidators[0].accumPower -= validator.power
  sortedValidators = sort(validators, fn(v){ v.accumPower })
```

From hereon, a proposer is chosen for each round according to:

```
// function definition
fn getNextProposer():
  proposer = sortedValidators[0]
  for each validator:
    validator.accumPower += validator.power
    proposer.accumPower -= validator.power
  sortedValidators = sort(validators, fn(v) v.accumPower )
  return proposer
```

Each consensus round is composed of two steps. Each of these two steps takes $T + i\delta$ where δ is some fixed increment of time, so each round is longer than the last one by 2δ. Any messages to be broadcasted are sent in the beginning of the step, and messages can be received in the background during the entire duration of the step.

```
// consensus round i for height h for validator V:
// (<message>)σ is a message signed by validator V.
// A <Proposal> is composed of a block, a blockhash for that block,
// and a list of <Vote>s from the previous round.

lastVote = nil
```

```
// step 1: Propose
proposer = getNextProposer()
if proposer == self:
  if i == 0:
    broadcast (<Proposal <Block h>, blockhash, nil>)σ
  else:
    broadcast (<Proposal <Block h>, blockhash, [<Vote>,...]>)σ
  wait for T+i*δ
else:
  receive <Proposal> for T+i*δ

// step 2: Vote
if no proposal was received or proposal is unacceptable:
  broadcast (<Vote V, h, i, lastVote>)σ
else if the proposal is acceptable:
  broadcast (<Vote V, h, i, blockhash>)σ
  if isDeciding(proposal, blockhash):
    decide(<Block h>)
receive others' <Vote>s for T+i*δ
```

A proposal is acceptable if the block is valid and either the round is 0 or the proof of acceptability (list of `<Vote>` messages from the previous round included in the proposal) consists of at least 2/3 of the voting power, where the vote value is either *nil* or equal to the block hash of the proposed block. Intuitively, a vote for *nil* is like a vote for any block. If at least 2/3 of the voting power in the proof of acceptability is explicitly for the proposed block (rather than *nil*), the proposal is deciding and the recipient *decides* on that proposed block by signing the block.

Lemma 2: If there are less than 1/3 in byzantine voting power & if at least one good validator decides on a block, then all good validators will decide on the same block eventually.

A good validator will only decide on a block if the block is explicitly voted for by at least 2/3 the voting power in the previous round. Since there are less than 1/3 in byzantine voting power, more than 1/3 of the voting power will remain true to the algorithm and continue to vote for the same block going forward, and will not change their vote to a different block unless it encounters an acceptable proposal for a different block. However, no proposal for a different block can be acceptable.

Lemma 3: If there are less than 1/3 in byzantine voting power, consensus is eventually reached (the algorithm terminates).

The proof is similar to the one in [2].

4.4 Committing to the agreement

While the consensus mechanism in the previous section gets good validators

to agree on the next block, we also need to ensure that validators stick to their commitments for blocks previously decided upon. We achieve this by incentivizing validators to sign the agreed upon block by rewarding them with transaction fees in proportion to their voting power, and strongly incentivizing validators to sign only one block at a given height. When signing a block, a validator must sign a string that includes the block's hash as well as the block height. When a block cheats by signing more than one block on the same height, a short *evidence* transaction can be included by anyone as long as it is committed before the cheater's bonded coins are released (after the unbonding period). When such evidence is found and committed, that validator's bonded coins get redistributed to the remaining validators in proportion to their voting power immediately.

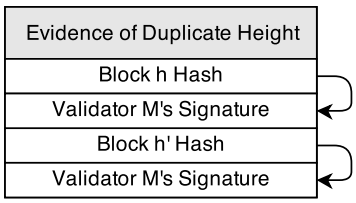

Figure 4: Evidence of duplicated height

As long as there are less than 1/3 in byzantine voting power, each successive block in the blockchain will have at least 2/3 in votes. Thus given a parent block, the correct child block is the one that has at least 2/3 in votes, and this uniquely identifies the current blockchain fork.

4.5 Cooperation

Since validators divide the transaction fees of *block h* amongst themselves, a greedy validator might be tempted to exclude some signatures when proposing the next *block h+1*. This is an inferior strategy when considering that other validators are game optimal participants. Given that the total amount of fees to be divided in a block is f_1, and that the sum of the voting powers v_i of all validators that have signed and successfully propagated their signatures is 1, consider proposer P with voting power $v_p < 1$ who is considering whether to include validator Alice's signature with voting power $v_a < 1$. At stake is Alice's fair share of the fees which is $f_1 \cdot v_a$. Of this, P's incremental benefit of excluding Alice's signature is:

$$f_1 \cdot v_a \cdot v_p/(1 - v_a)$$

Then, *Alice* could react tit-for-tat by excluding P's signature when it becomes *Alice*'s turn to propose the next block, where the sum of the fees in that block is f_2. In that case, P's detriment is:

$$f_2 \cdot v_p$$

P only gains a monetary advantage if the benefit outweighs the costs where:

$$f_1 \cdot v_a \cdot v_p/(1 - v_a) > f_2 \cdot v_p$$

$$f_1 > f_2/(v_a/(1 - v_a))$$

Thus if P and *Alice*'s interactions were limited such that they only get to propose one block each, it's clear that P doesn't benefit overall unless the proposed block contains a much larger sum of fees f_1 in reward than what *Alice*'s later block will contain, f_2, assuming $v_a \ll 1$. Even if *Alice*'s voting power is large, she could divide her stake amongst multiple smaller accounts. In the case where P and *Alice* aren't limited to propose one block each, P and *Alice* might exclude each other's signatures indefinitely. In this case, P's expected benefit on each block is:

$$E[fees] \cdot v_a \cdot v_p \cdot v_p$$

whereas P's expected detriment on each block is:

$$E[fees] \cdot v_p \cdot v_a$$

No matter the amount of voting power, no two validators benefit by excluding each other's signatures indefinitely. Intuitively, this is because the other validators gain more when two validators exclude each other.

5. Checkpoints

While the protocol described so far is theoretically feasible, in practice there are computational, storage, and network limitations to consider. We want to allow for as many validator nodes as possible, but it may be too costly to store every validator's signature for every block. For a concrete example, we estimate the total number of unique active miners in Bitcoin to be on the order of 50,000. If every validator signed every block and consensus was reached on average every minute, and each signature were 32 bytes long, that totals to 840 Gb of storage every year just for validator signatures. For this reason we propose a checkpointing system such that the validator signatures of most blocks can be pruned away.

Every 240 blocks a checkpoint is created that summarizes the voting activity of each validator. The checkpoint data structure is hashed into a merkle tree and the root hash is included in the next block. Each row in the checkpoint data structure corresponds to a validator, ordered by the age of the validator. In the figure below, *validator #4* has been offline and not signing anything.

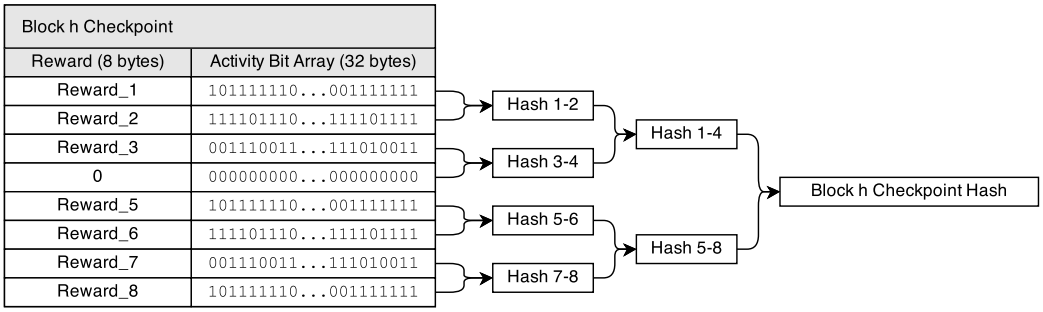

Figure 5: Checkpoint merkle tree

When a validator signs an invalid checkpoint, evidence is included in the blockchain, and its bonded coins get redistributed to the remaining validators in proportion to their voting powers. The evidence for signing an invalid checkpoint is as follows:

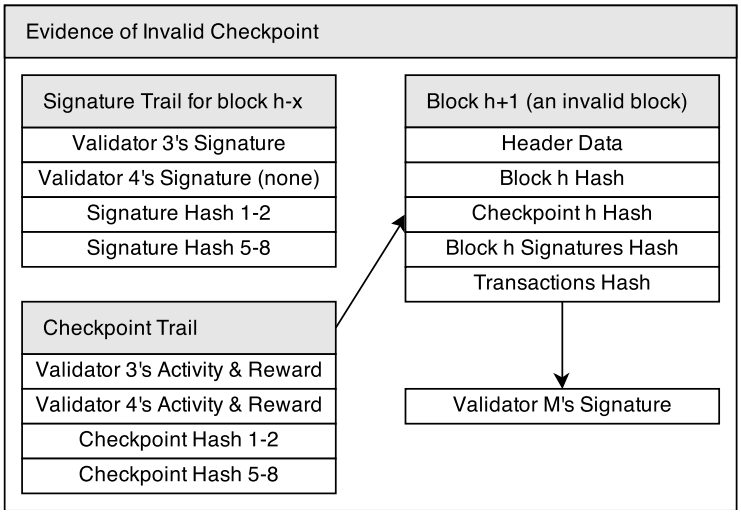

Figure 6: Evidence for signing an invalid checkpoint

Note that the size of the evidence of a validator having signed an invalid checkpoint is *O(log(len(validators)))*. Validator signatures for blocks older than the last checkpoint (240 blocks) can be pruned as long as there are some validators that keep all the signatures of the last X blocks, where X is the unbonding period. These validators are incentivized to keep the older signatures in order to be able to produce such evidence.

6. Notes

While our algorithm achieves consensus at every block given our assumptions, it is still possible for a set of validators that collectively had more than $2/3$ of the voting power at some point in the distant past (but has since unbonded their

coins) to create an alternative fork of the blockchain. Thus we require that the unbonding period X be long enough for users (who aren't actively participating in the consensus mechanism) to periodically reconnect to the network at least once every X blocks, or to place trust in a set of nodes of the network that can identify a valid checkpoint within the past X blocks. New users should also be seeded with a trusted checkpoint, else they are also vulnerable to being duped by a malicious fork of the blockchain.

References

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

2. C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM*, vol. 35, no. 2, pp. 288–323, 1988.

3. M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, 1985.

4. M. Correia, G. S. Veronese, N. F. Neves, and P. Verissimo, "Byzantine consensus in asynchronous message-passing systems: a survey," *International Journal of Critical Computer-Based Systems*, vol. 2, no. 2, pp. 141–161, 2011.