# TenderMint: Consensus without Mining

Jae Kwon yk239@cornell.edu Draft v.0.5

**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 proofof-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 process, 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 DLS (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  $\Delta$  on the time of messages to be delivered. Intuitively, there may be arbitrary but finite latency in the network. We also assume that all nonbyzantine 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. As in the algorithm proposed by Dwork et al, it can tolerate 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 process begins for the next block at height h.

Lemma 1: The consensus process for block height h for all good validators begins within  $\Delta$  of each other in global time.

The proof follows from the definition of  $\Delta$ . 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  $\Delta$ .

Let T be some fixed duration of time that is suspected to be at least  $2\Delta$ . We don't know what  $\Delta$  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 process begins by first waiting W, then proceeds in rounds until consensus is reached.

At each consensus round, we use a deterministic algorithm based on the prior blockchain history to compute a proposer; proposers are chosen in proportion to their voting power.

```
// First, copy AccumPower over to RoundAccumPower
for each Validator {
    Validator.RoundAccumPower = Validator.AccumPower:
7
// Determine proposer for the next round
function getNextProposer():
    // Increment voting power
    TotalIncremented := 0
    for each Validator {
        Validator.RoundAccumPower += Validator.Power;
        TotalIncremented += Validator.Power
    }
    // Determine the designated Proposer for this round
    Proposer := Validator with the most RoundAccumPower;
    // Decrement from Proposer and make Sum(RoundAccumPower) zero.
    Proposer.RoundAccumPower -= TotalIncremented;
    return Proposer;
}
```

Once a block is committed we increment Validator. AccumPower similarly.

```
// Increment voting power
TotalIncremented := 0
for each Validator {
    Validator.AccumPower += Validator.Power;
    TotalIncremented += Validator.Power
}
// Determine the designated Proposer for round zero
```

```
// Decrement from ProposerR0 and make Sum(AccumPower) zero.
ProposerR0.AccumPower -= TotalIncremented;
// Adjust Validators membership and other state data
// based on the Block we're committing
...
```

Lemma 2: For a given block height and a round of the consensus process, all good validators agree on a proposer for that round.

Since the previous block was signed by more than 2/3 of the voting power and less than 1/3 of the voting power is byzantine, no other block could have received 2/3 of the voting power. This implies that all good validators start the consensus process off of the same blockchain. The sequence of proposers is completely determined by a common blockchain, so all must agree on the sequence.

Each consensus round is composed of three steps. Each of these three steps at round R takes  $T + R\delta$  where  $\delta$  is some fixed duration of time, so each round is longer than the last one by  $3\delta$ . 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 rounds at height H
// <Message>\sigma is a message signed by validator V.
LockedProposal := nil;
for consensus round R {
    // Step 1: Propose
    Proposal := nil;
    if getNextProposer() is me {
        if LockedProposal != nil {
            Proposal = LockedProposal;
        } else {
            Proposal = constructProposal();
        }
        broadcast(<Proposal>\sigma);
    } else {
        Proposal = getBroadcastedProposal();
    }
    // Step 2: Vote
    if LockedProposal != nil {
        broadcast(<Vote V, H, R, LockedProposal>\sigma);
    } else if Proposal is valid {
        broadcast(<Vote V, H, R, Proposal>\sigma);
    }
    Votes = getBroadcastVotes();
    // Step 3: Pre-Commit
```

```
if a <sup>2</sup>/<sub>3</sub>_majority(Votes) exists {
    // Set lock
    LockedProposal = 2/3_majority(Votes);
    broadcast(<Precommit V, H, R, LockedProposal>\sigma);
}
Precommits := getBroadcastPrecommits();
// End of Step 3: Commit and Unlock
if <sup>2</sup>/<sub>3</sub>_majority(Precommits) exists {
    nextBlock := <sup>2</sup>/<sub>3</sub>_majority(Precommits)
    // A Commit by validator V at height H round R
    // sets all Votes and Precommits for future rounds
    // at height H round R+1, R+2, R+3, ...
    broadcast(<Commit V, H, R, nextBlock>\sigma);
    commit(nextBlock);
    break; // Consensus complete for height H
} else if <sup>1</sup>/<sub>3</sub>_majority(Precommits) exists which isnt LockedProposal {
    // My LockedProposal is outdated, unlock!
    LockedProposal = nil;
}
```

Lemma 3: If there are less than 1/3 in byzantine voting power & at least one good validator decides on a block B, then no good validators will decide on any block other than B.

This is the safety guarantee. Consider the earliest round R where at least one good validator commits block B at round R. We know that more than 2/3 of precommits were for block B at round R. Considering that less than 1//3 are byzantine, we know that at least 1/3 of good validators must have pre-committed block B at round R. These validators must have a lock on block B at round R. No other block can be committed by good validators in rounds after R unless some of the good validators unlock from B. The only way these validators can unlock from B is if more than 1/3 pre-commit for a block other than B after round R. This implies that at least one good validator pre-committed a block other than B after round R, which can only happen if more than 2/3 voted for a block other than B after round R, which is impossible.

Lemma 4: If there are less than 1/3 in byzantine voting power, consensus is eventually reached (the algorithm terminates).

TODO: Prove liveness. The proof is similar to the one in [2].

4.4 Committing to the agreement

}

While the consensus process 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 outweight 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. Validator Signature Compression

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.

TODO: Describe cryptographic multisignatures.

#### 6. Changing participation

So far we have considered mute validators to simply be byzantine, but ideally the network adapts in response to validator churn. Validators that haven't signed for Y blocks in a row (where  $1 \le Y \ll X$ ) are considered to have timed-out and are *implicitly unbonded*. As long as validators churn in a staggered fashion, the *active* validator set can adjust to account for a changing set of participants. Validators that need to go offline temporarily should be incentivized to sign an *explicit unbonding* transaction as opposed to timing out, as that allows the active validator set to adjust immediately. For example, timeouts could result in a some small penalty fee in proportion to the amount of bonded coins, or validators may need to participate for some number of blocks before earning any fees. Later, (both explicitly and implicitly) unbonded validators can become active once again by signing a block agreed upon during a consensus process by the remaining active validators, before the unbonding period is over.

## 7. Security

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 with no recourse of retribution by the protocol (since the unbonding period had already passed). Such a gratuitous (sham) blockchain fork must have a most-recent-common-block that is at least X blocks ago. Thus we require that the unbonding period X be long enough for users (who may not be actively participating in the consensus process) to periodically reconnect to the network at least once every X blocks, or to place trust in a set of persistent nodes of the network that can identify a valid checkpoint within the past X blocks. Alternatively, users may place trust in a set of known validators to participate for an extended duration of time; if any validator signatures in the trusted set aren't present, it signals the user to perform a manual check when resyncing with the network.

This is a significantly different trust model than other protocols that require persistent trust in a centralized checkpointing authority. In proof-of-work based consensus protocls such as in Bitcoin, once a user has downloaded a trusted client, the user can wait indefinitely before connecting to the network for the first time or reconnecting after an extended period of absence. In contrast, a TenderMint user must have a trusted client and recent checkpoint when connecting to the network for the first time or after an extended period of absence; once connected, the user need not trust any authority for as long as the last synchronized block is recent (within X blocks of the current block of the network). In practice this isn't an onerous requirement as long as the unbonding period is long enough: say, a year.

## 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.
- 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.
- 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.