

Forum: [Hack Forums](#)

Thread: [Help with new consensus model?](#)

jkjoelbee - February 5, 2025, 12:34 pm

I'm working on a new cryptocurrency that uses the recently NIST standardized quantum-safe signature scheme, CRYSTALS-DILITHIUM at NIST level 5. The produced signatures are roughly ~5,000 bytes compared to ECDSA signatures at 64 bytes which causes a lot of unique bottlenecks for current consensus models, mainly regarding network bandwidth. There are technically other quantum-safe signature schemes (non-standardized) that offers slightly smaller signatures, but still at least 10x larger than ECDSA signatures.

As a quick reference, if Bitcoin were to use quantum-safe signatures that were 2,000 bytes each, throughput of the network would be reduced from ~7 transactions per second to ~1 every 2 seconds. If other networks were to use quantum-safe signatures, their throughput would also be reduced in the same manner without significant changes to the protocol, which is the big issue with quantum-safe signatures (not to mention how a transition for a current cryptocurrency would be pretty much impossible without a major hard fork and community participation).

So where I'm hoping I could potentially get some insights is the actual consensus model itself, which is inspired by IOTA and the Lachesis protocol. This consensus model uses asynchronous validation and inferred voting in order to reach consensus, and I'm wondering what you guys think and how it could be made better. As it stands now, I think the inferred voting in this model is a great foundational idea, but the execution is not so great. Here's how it works.

This consensus model and ledger function essentially as a graph database, with each node representing a column, or in other words, a chain. This model relies on the following concepts to achieve consensus on any given transaction.

Proof-of-Tenure: Instead of using a system like Proof-of-Stake to prevent Sybil attacks and weigh the voting process, this consensus model uses Proof-of-Tenure. This allows anyone to participate in the consensus process without an upfront investment while maintaining security through a weighted tenure system. Good actions and uptime equate to a higher tenure score and selection weight, while bad actions and downtime result in a lower tenure score and selection weight.

Asynchronous Validation: Nodes can achieve consensus on multiple transactions simultaneously, process "out-of-order" transactions, and validate transactions even in a high-latency environment—all while achieving fast finality.

Inferred Voting: In typical consensus models that aren't Proof-of-Work, there are usually "consensus rounds" where the network or a "committee" exchanges votes to reach consensus on transactions, consuming significant bandwidth. With inferred voting, this consensus model reduces up to 95% of extra network overhead while improving scalability by including "votes" (references) in all network interactions instead of explicitly exchanging messages for votes.

Phase I: User Submits a Transaction

A user submits a transaction to the network, which is deterministically routed to a specific node based on the transaction hash.

The node verifies the transaction (signature, addresses/username, balances, timestamp, etc.).

The node begins forming an "event block" with the transaction to add to its own chain.

The node randomly selects two node chains and their latest event blocks and references them along with its previous event block.

The node signs the completed "event block," appends it to its chain in the database, and

propagates the event block to the network. Phase II: Nodes Receive the Event Block

Nodes receiving the event block verify the transaction inside it (signature, addresses/usernames, balances, etc.) and also verify the event block as a whole (node signature, references, etc.).

They append that event block to the appropriate node chain in their local copy of the ledger.

Phase III: Consensus and Finality

For consensus and finality to be achieved for a specific transaction, more than 50% of the current tenure weight of the network must reference that event block or an event block that came later in the same node chain.

This enables inferred voting between nodes without direct communication of votes, reducing network bandwidth usage while making this consensus model as scalable—or even more scalable than current cryptocurrencies. It achieves this while remaining fully decentralized and supporting larger quantum-safe signatures. By referencing other event blocks from separate node chains, a node effectively votes for the validity of that event and all prior events in that chain.

What happens if the network is inactive for a while?

Since transactions are confirmed by other transactions, the confirmation time can become unstable if activity is low. To keep confirmation times steady, nodes can create empty event blocks every few seconds if there are unconfirmed transactions. However, as the number of nodes grows, the time between these empty event blocks increases to prevent network overload. For example, if there were a million nodes and a one-second interval, half a million empty blocks could flood the network. But with only 100 nodes, a one-second interval would be fine.

What if the node responsible for a transaction is offline?

If the node assigned to process a transaction is offline, the network keeps trying to send it to that node. If the node comes back online, the transaction proceeds as usual. If not, the transaction expires after 60 seconds, and the user must resubmit it. Nodes that fail to process transactions lose tenure and may even be removed from the network, requiring manual rejoining. Transactions always expire in 60 seconds unless fully confirmed before that time.

Should more than two references be used in an event block?

Yes, typically around 25 references per event will be used. Each extra reference adds very little computation but requires about 32 extra bytes of data.

How does the network handle splits or partitions?

The system is designed to avoid network splits, but if one occurs, usually only one version of an event is confirmed while the other expires. In some cases, both/all versions expire.

How does this consensus model prevent double spending?

Honest nodes won't confirm invalid transactions. If someone tries to double spend and only has enough balance for one transaction, the first will likely be confirmed while the second is rejected and expires.

Here's some of the issues I'm having. Please let me know what you guys think and how this could be made better.

I'm aware of the issue of deterministically routing a transaction to a node based on the hash. This makes network splits much harder to happen, but if that node goes offline, the user who submits a transaction will have to wait for the transaction to expire before resubmitting (or risk both going through). How could this be reworked to not affect the user experience negatively?

How would distribution/minting work?

The current thoughts are a node will have to post their own reward (deterministically calculated) to their own node chain every X amount of minutes, but not sure if this is the best way to go about it.

Another thought I had is have X amount of the crypto be minted during every transaction, but that would be very unstable.

Overall, I like the idea of this consensus model, but can't work out the details quite yet. Any help is appreciated.

jkjoelbee - February 5, 2025, 1:07 pm

(Feb 5, 2025 01:49 PM)Lowkey Wrote: [CITE](#) How much do you know about this already?

You're saying the nodes post their own rewards every X amount of minutes but that just creates instability or unfair distribution no?

Instead of minting per transaction (i know you said minutes but that could actually be kinda just...whenever right?)

What if you implement a network-wide adaptive issuance schedule?

Basically new coins are minted at a predictable rate but this time distributed based on node tenure and participation.

Also you have a potential delayed finality issue.

That inferred voting mech is cool for reducing bandwidth but could fr be a problem in the future. To answer the first question, I guess it depends on what you're specifically referring to and what the frame of reference is haha.

About the minting and distribution, those were two separate examples. I've implemented both already and don't really like either one, but can't think of another low resource way to do it. The minting per transaction was one of the first ideas, but definitely leads to instability with supply and minting schedule. The current method implemented is essentially have a total network reward for all active nodes that are distributed every 60 seconds (and that reward is quartered every 2 years until supply cap is reached). Each node would deterministically calculate their own reward and then post it to their event chain, which is then propagated. The actual distribution amount per node for that 1 minute time frame would be determined from their tenure score. Is this essentially what you proposed for a "network-wide adaptive issuance schedule"?

And yes the potential delayed finality is also an issue I'd like to fix, although it's not a "big" issue per say. As it stands with a small test network, that's not really an issue at all with such a small number of nodes and the empty event block "fail safe" (finality in under 1 second), but could potentially become an issue with a larger amount of nodes such as 10,000+. The finality time in the currently implemented way would be predictable based on the amount of nodes in the network, but could be seen as "unstable" since it could change on a day-to-day basis.

Thank you for your input.