PBFT-based consensus protocols provide some good properties such as fast finality over chain-based protocols (e.g. Bitcoin, Cardano Ouroboros), but they typically don't scale well with the number of nodes participating in the consensus. This is because PBFT-based protocols require all-to-all votings to reach consensus (unless probabilistic tricks like Algorand-style committee sampling is in use). Even with latest proposals like HotStuff which avoids the complexity in view changes, each consensus decision still requires O(N^2)

voting messages flowing through the network, where N

is the number of nodes. Imagine a system with 100k nodes, in each of the consensus voting phases, every node has to receive and process 100k voting messages from other nodes. This could overwhelm the network, and thus limit the number of participants in PBFT-based protocols. This makes PBFT-based protocols less decentralized compared to chain-based protocols.

To address this bottleneck, we propose to leverage SNARK proof aggregation to reduce the amount of voting messages. Recently, @vbuterin wrote a nice introduction to SNARK proof aggregation/merging.

The basic idea is to use SNARK aggregation in a way similar to signature aggregation, so that an aggregated SNARK can compactly encode which nodes have voted for a block. For simplicity, let us assume the network has a fixed number of N

nodes. Assume a new block has just been proposed, and the i^{th}

node n_i

wants to vote for the block. It does the following:

- First generate a SNARK proof \pi_i

representing his yes vote (basically it just needs to create a SNARK proof \pi_i

for his signature \sigma = sign_{sk}(h)

, where h

is the hash of the new block).

- Broadcast the SNARK proof \pi_i

, and an index vector

v_i

to all its neighbors. The index vector is an N

dimensional boolean

vector that encodes which nodes have provided SNARK proofs for their yes votes. Initially, v_i = (0,0, ..., 1..,0)

, i.e., all but the i^{th}

element are zeros, since only node n_i

has provided the proof for its yes vote.

Meanwhile, Node n_i

keeps aggregating the proof and index vector and then broadcast them out to all neighbors:

- Upon receiving (\pi_j, v_j)

from its neighbor n_j

, it first verifies the validity of the proof verify(\pi_j, v_j) == true

. If so, it updates its local proof and index vector by \pi_i = Aggregate(\pi_i, v_i, \pi_j, v_j)

, v_i = v_i

OR v_j

. Here OR is the element-wise boolean "or" operator. The node stops doing aggregation when the number of "1"s in v_i

is over a certain threshold, e.g. \frac{2}{3}N

.

- Node n_i

maintains a local timer. Whenever the timer is triggered (e.g. once every 100ms), node n_i

broadcasts out the latest \pi_i

and v_i

. Note that in the first few rounds, most of the elements of v_i

are zero, so data compress can be applied to effectively reduce the message size.

The readers might wonder why we don't just use signature aggregation. The issue there is that signature aggregation typically requires v_i

and v_j

to be disjoint, i.e., the same element (e.g. the k^{th}

element) cannot be 1 in both vectors. SNARK aggregation overcomes this restriction.

It can be shown that in a gossip network with relatively good connectivity, only O(logN)

broadcasting rounds are required for the above voting process to converge (similar to how a gossiped message can propagated through the network in O(logN)

time). Moreover, since a gossip network has O(N)

number of edges, the total number of voting messages is O(NlogN)

, which scales much better as N

grows. The above process also has good tolerance to Byzantine nodes, since Byzantine nodes cannot forge fake proofs.

Is this a viable approach? Feedback and comments are appreciated!