RISE 1.3.0 Consensus change

Andrea B, Matteo C.

This document will introduce 4 new breaking changes that we'll deploy in RISE starting from version 1.3.x.
All the 3 consensus changes will be deployed in mainnet before the end of the year as long as community does not provide valid arguments against one or more of the 3 proposed changes.

1. Standby Delegates incentives

Current dPoS system allows only a specific number of fixed delegates. Delegates are chosen by sorting them using a ranking factor which usually boils down to the "Cumulative weight of votes received".

Definitions & Assumptions:

- Let `n` be the Number of Delegates that forge in every round
- Let `m` be the number of Delegates from which the `n` delegates are chosen from
- Let `W(d)` be the weight of delegate `d`.
- Let `R(d)` be the rank of delegate `d` using the `W(d)` as a sorting mechanism
- Let `P(d)` be the Probability of delegate `d` to have a slot in the current round.

Current implementations

- $n = m$,
- $P(d) = \begin{cases} 1, & \text{if } R(d) \leq n \\ 0, & \text{otherwise} \end{cases}$

Problems

- There is no immediate incentive for StandBy delegates;
- StandBy delegates can not actively participate in the network;
- Active Delegates are not incentivized in obtaining more votes;

- Generally a relatively high gap exists between the last active and the first standby delegate.

Proposed solution:

Instead of having `m=n` we propose to set `m` to a greater value and pick the `n` delegates out of `m` set so that the chances for a delegate having $R(d) \leq m$ of being chosen as forger in round `r` is proportional to its $W(d)$ and not fixed as in the current implementation.

Mathematically defining the probability of being chosen is not straightforward. For now let's just define the expected outcome for $P(d_x)$ which is:

$$P(d_x) = \begin{cases} \lfloor 0:1 \rceil, & \text{if } x \leq m \\ 0, & \text{otherwise} \end{cases} and\, P(d_x) \geq P(d_x + 1) \forall x$$

Example:

> m=202, n=101.
> Every delegate having $R(d) < 202$ could be chosen in the next round

Complicances and proposed algorithm

Randomly choosing $\binom{m}{n}$ out of a weighted set of delegates is not an easy task.

Assumptions & definitions

- Let's define $D(m) : d_i \in D \mid R(d_i) < m$ as ordered by rank delegates from rank `1 to m`
- So that:
  - $|D(m)| = m$
  - $R(d_i) < R(d_{i+1}) \forall i$
  - $W(d_i) \leq W(d_{i+1}) \forall i$
- Let's define $S(D(m), r_i)$ as the randomly sorted list of delegates in $D(m)$ using $W(d)$ for round `i`.
- In order to reach consensus, every node should independently compute the same $S(D(n), r_i)$.
- **Note**: $|S(D(m), r_i)| = n$

Hence we need to find a way to sort $D(m)$ using $W(d) \, \forall \, d \in D(m)$. Such problem is called `Weighted Random Sampling` and a possible $O(n \log_2 n)$ was provided by

Efraimidis, Spirakis in 2005. (Further optimizations are possible to lower complexity up to $O(m \log_2 \frac{n}{m})$.

Algorithm

Input: D(m), r_i
Output S(D(m))

1. $tmpSD : WRS(D(m), n, seed(r_i))$
2. $return : shuffle(tmpSD, seed(r_i))$

Caveats

1. Both steps needs a random number generator. One of the requirements was a stable output. In order to achieve that we will need to use a stable seed for the *RNG*.
2. When Applying $WRS()$ over $D(m)$ we need to make sure to sort the elements using an extra factor to ensure there are no ambiguous results over elements having the same weighted result. We propose the use of delegate `publicKey` for this purpose.

Random seed

As previously stated the random seed needs to be carefully chosen. The easier possible random seed would be using the round index.
But, using such seed will give the whole network the ability to precompute $S(D(n))$ and try to cheat by slightly changing their rank position.

Let's define:

- $shm(r)$ as the `height` at which **at least one network** participant knows the seed for Round `r`.
- $shM(r)$ as the `height` at which the **majority** of network participants knows the seed for Round `r`.

Then: $shm(r) \leq shM(r) < firstBlockIn(r)$

Example. When using `r` as parameter then $shm(r) = shM(r) = 0$ since all the seeds can be precomputed since genesis.

A safer approach would be using the `lastBlockIn(r)` computed `ID`. This will produce the following effect:

- $shm(r) = lastBlockIn(r-1).height - 1$
- $shM(r) = lastBlockIn(r-1).height$

The reason for $shm(r)$ value is because the `lastBlockIn(r-1)` forger will know the seed before the block is actually accepted by the network. Possibly, the last delegate in each round, could include a cheap transaction that will serve as `nonce` to change the `ID` until the generated seed solves his agenda.

Removing cheating incentives

To discourage last round forger to misbehave, we will **exclude him from the next round** effectively removing any of his incentives to cheat.

To rebalance the newly missed round we'll artificially set the last block forger in a round to the one who forged a last block in around less recently.

This will also **improve decentralization** as we can be certain that delegates in `Round+1` are not the same as the delegates in `Round` (or `Round+2`).

Pros

RISE has 101 active delegates and ~97% of the voting power is spread among the top 101 delegates leaving no room for a new standby delegate to join the active delegates. At the time of writing there are about 110 nodes running the mainnet core clearly proving that standby delegates has no real incentive in maintaining a node running.

By picking the next 101 forgers from a pool of, let's say, 150 delegates; all the delegates in such ranking range will get a chance to forge a block hence providing incentives to maintain a node.

Furthermore, community will be more inclined to vote for a new promising delegate as they'll instantly increase the chance for the newcomer to forge in the next round **allowing him to prove himself**.

2. Productivity as a ranking factor

In RISE a delegate could miss several blocks and not get penalized for his poor commitment.

What we propose is to use a mechanism already adopted by Adamant[2]. Such improvements slightly change the votingWeight formula by also multiplying the result with $\frac{pB}{pB+mB}$ with some caveats. Note: `mB =>` missedBlocks, `pB =>` producesBlocks

The resulting voting formula will be something like

$$w(d) = votingWeight(d) \cdot \begin{cases} 1, & if\ mB(d)\ +\ pB(d)\ \leq\ 200 \\ \frac{pB(d)}{pB(d)+mB(d)}, & otherwise \end{cases}$$

Example: Given a delegate with a votingWeight of 100, 50 missed blocks and 950 produced blocks, the $w(d)$ will evaluate to = 95.
Example 2: 100 missed blocks, 99 produced blocks and 60 votingWeight will still result 60 till the next block is either produced or missed.

Immediate effects on RISE Mainnet

There are several uncaring delegates that did not upgrade their node and are not forging for months. They'll see their voting weight reduced and their rank along with it.

Long run effects

We expect a much more prompt response to network changes, forks and node updates once this hits mainnet.

3. Fair votes system

The problem

In multi vote DPoS blockchain where voting weight is not split among voted delegates there is a high risk for organized groups and cartels to take over the blockchain.

A solution

We think that if an account votes for multiple delegates at once, then its voting weight should be proportionally split among all voted delegates.
As of now, if an account with 100 RISE votes 2 different delegates they will both get 100 worth of weight.
As you may know RISE only allows 1 vote so this improvement won't likely cause any

direct effect on RISE.
We need to keep in mind that RISE aims to be a blockchain platform where external developers can decide their own logic.

Every time we, at RISE, decide to change something we have to carefully evaluate possible butterfly effects[3] over future sidechain implementations built upon RISE.

4. Delegates Banning

The problem

As previously stated, RISE Mainnet currently has several delegates who did not update their servers and do not care about maintaining their nodes.

As a result these uncaring delegates are effectively hurting the network.

Unfortunately, even with the previous improvements, these delegates will probably always have a chance of being picked as one of the next forgers.

For this reason we're going to introduce a new delegate banning mechanism.

A solution

If a delegate misses enough blocks **in a row**, the network will automatically filter him out effectively assigning him a probability equal to zero of being chosen amongst the next round forgers.

We believe that allowing 3 days for a delegate to restore their node could be a good tradeoff.

When a delegate is banned from the network there is no turning back. He will never be able to forge again! However, he is allowed to create another account and register a new delegate; he will also need to ask his previous voters to support him proving he resolved the solution.

Conclusions

We believe that enabling this now will be essential in the very near future of RISE.

We invite everyone who wants to participate the discussion to join our slack.

Changelog

- *v2*: 2018-11-16:
  - Added Peer banning
  - Changed anti-cheat policy for last round forger
  - Simplified math proposed solution
- *v1*: 2018-11-01