This post is a proposal on the future of POS on Ethereum. My skills in programming such a network being limited, I won't go into too much technical detail. On the other hand, I will be interested in feedback on the technical feasibility of this proposal as well as the risks of new attack vectors that this could introduce.

It may not be all good to keep, but my goal is to start thinking about how to anticipate the future of Ethereum. An outside point of view can bring interesting things. I do not believe that the elements mentioned in this proposal are urgent for the proper functioning of the network.

Also, English is not my native language so excuse me for my possible mistakes.

Introduction

Reading @vbuterin Simplified Active Validator Cap and Rotation Proposal, it seemed to me like a good opportunity to improve the health of the network by simplifying downtime management.

Indeed, one of the duties of a validator is to keep his node operational 24/7. This is important for the finalization of the network. If we look at the health of the chain after 5 months, the participation rate is excellent so far (~ 99%). This shows that the 125K validators manage to accomplish their mission.

On the other hand, there were nearly 135validators slashed. Most of the time, these validators had configured a second machine to take over if the first happened to stop. This method has been strongly discouraged

since launch, yet it represents the majority of the penalties (although the number of slashing events seems to decrease over time which shows the growing maturity of the network.)

Over time, the complexity of running a validator is likely to increase due to:

- Sharding
- Increase in the number of validators (this proposal helps to solve that)
- Size of the blockchain (stateless is a solution to this problem)
- Other unknown evolutions

Moreover, in the event of an increase in the value of ETH, it is reasonable to upgrade your machine to increase its safety and reliability.

@vbuterin has proposed in his article a cap to reduce the load on the network if the size of the active validator set is too big. This helps to slow the growth in the number of validators when it is sufficient for the network to be decentralized and functioning properly. Developers' work is thus simplified since there is no need to unnecessarily overdevelop clients to make them work with several million of peers.

The system of rotation between validators also makes it possible to treat all validators equally regardless of their age. If a hard cap was imposed, the validators already present in the network could remain there indefinitely and block the arrival of a new validator. I think it would be a risk for the good functioning of the blockchain to give power to a group of validators . Decentralization is one of Ethereum's priorities and I don't find having a closed validator group fits with that spirit.

A cap also discourages too much liquidity from being blocked to secure the network when not needed. This without changes the insurance curve to reduce the gains.

Vitalik's proposal enables returns to be reduced in proportion to the number of validators.

Simplified Active Validator Cap and Rotation Proposal

Once the active validator set size exceeds MAX_VALIDATOR_COUNT

, validator returns should start decreasing proportionately to 1/total_deposits

and not 1/sqrt(total_deposits)

.

This will reduce the interest in staking, but you can end up in a situation where the returns are still sufficient for a large amount of ETH to be staked but inactive and therefore useless.

My idea would be to slightly

modify this system to allow the validators to put themselves to sleep and this would have the effect of improving the health of the network at the same time.

Proposal

In order to make this cap useful to all network actors, validators should be given the opportunity to “fall asleep

” voluntarily.

The network will set a target on the number of validators, let the validators manage themselves and only intervene in the event of an imbalance.

How could this work?

Let’s take the constant MAX_VALIDATOR_COUNT

from the Vitalik’s proposal representing the sufficient number of validators necessary to operate the network in a decentralized manner without being too burdensome for clients.

We will also keep the constant ROTATION_RATE

which represents the maximum fraction of the network renewed each finalized epoch.

Now let’s add the element that will evaluate the different validators :

N_Reward_Score

(NRS

) which corresponds to the sum of a validator’s rewards over the last N epochs. Again, this number N must be determined to provide a sufficiently long period to correctly evaluate a validator.

In order to delay the system and leave the final decision to the protocol, it is necessary to set up two queues. One to put the validator to sleep called the Sleeping Queue

(SQ

) the other to wake it up call Wake up Queue

(WQ

). A sleeping validator is not protected from penalties for past bad behavior. But it no longer receives attestation work or proposals so it also no longer has any rewards.

Name	abbreviation	note
MAX_VALIDATOR_COUNT		maximum number of validators
Active_Validator_Count		number of active validators
ROTATION_RATE		maximum fraction of the validator set change per epoch
N_Reward_Score	NRS	sum of reward last N epoch
Sleeping Queue	SQ	queue to put a validator asleep
Wake up Queue		

WQ

queue to wake a validator up

So all the elements of the system are in place for it to work.

The rules are as follows :

if Active_Validator_Count

< MAX_VALIDATOR_COUNT

in need of validators case

WQ

empty

wake up the sleeping validator with the lowest NRS

WQ

not empty

wake up one validator in WQ

if Active_Validator_Count

MAX_VALIDATOR_COUNT

to much validators case

SQ

empty

put asleep the active validator with the highest NRS

SQ

not empty

put one validator in SQ

asleep

When we reached equilibrium :

if Active_Validator_Count

== MAX_VALIDATOR_COUNT

SQ empty

SQ not empty

WQ empty

do nothing

wake up the sleeping validator with the lowest NRS

and put one validator in SQ

asleep

WQ not empty

put asleep the active validator with the highest NRS

and wake up one validator in WQ

put one validator in SQ

asleep and wake up one validator in WQ

This happens as many times per epoch as the ROTATION_RATE

allows.

The protocol will automatically force the validator that is too much active (high NRS

) to sleep and it will also wake up any inactive validator if its NRS

is too low.

I haven't discussed how the protocol chooses which validator to take in each queue. I have 2 possibilities: either simply by FIFO or then classified each validator in queue by its NRS

. (This element must be studied to determine the best solution.

)

As we can see, when the count is unbalanced and there is not enough validators in queue, the protocol can force them to "wake up or go to sleep

".

A dormant validator, especially if their NRS

score is low, should always be ready to be awakened.

If a validator is asleep, its NRS

will go down and so its NRS

ranking in the inactive validator set.

The opposite occurs for an awakened validator.

First advantage : managing downtime

Before going any further, let's already look at the first advantage which is to simplify maintenance operations.

If validator needs to perform maintenance or an update, he can anticipate it by being more active to increase his NRS

and then put his validator to sleep without the protocol waking him up because of a too low score.

This will give him time to perform the maintenance without downtime either for him or for the beaconchain.

Likewise, if a period of risk for the validator is approaching, such as for example a planned power cut or connection loss, or even periods of absence which make it impossible to intervene in the event of a problem, the validator will be able to accumulate rewards then place is validator to sleep to avoid risks.

In the case where Active_Validator_Count

== 2 * MAX_VALIDATOR_COUNT

. a validator will have to spend an average of 1/2 of their time asleep. He will therefore have plenty of time to maintain his validator.

In addition to that, it would then be possible to include a security in the client such as if the validator fails X consecutive attestation or X block proposal then place the validator in the

SQ

This allows time for the validator experiencing difficulty to react and correct the problem. In addition, this will improve the health of the network by removing fast a faulty validator to replace it with a functional one. And in the event of a bug on a large number of validators, they will quickly fall asleep to be replaced.

To go further 1: the red zone

This last point aiming to put a validator to sleep automatically could be implemented directly in the blockchain. A validator who does not meet its duties can be automatically placed in sleep mode by the blockchain. This solution will therefore be faster to remove an inactive validator than to subject it to inactivity penalties.

But how do you exit a validator that is no longer active if it’s asleep?

I propose to implement a penalty system for sleeping validators with the lowest NRS

. A threshold will be established to distinguish the less active validators. It can be chosen as a percentage of sleeping validators or proportional to the ratio Total_Validator_count /MAX_VALIDATOR_COUNT

. (The fairest solution should be studied to penalize only inactive validators.

)

If the network is in lack of validator (if Active_Validator_Count

< MAX_VALIDATOR_COUNT

and WQ

empty) instead of waking up the sleeping validator with the lowest NRS

, it will wake up the validator with the lowest score but above the threshold of the red zone

. (To not introduce an inactive validator in the active set)

Validators in the red zone

will suffer a inactivity leak until their balance drops below 16 ETH and they are thus kicked from the network, or they can request to be placed in the WQ

to exit the red zone

. The validator will therefore have to act to be active again. The beaconchain can exit inactive validators without penalizing its own proper functioning.

To go further 2: exit and partial withdraw

Having some validators asleep can also facilitate the exit and partial withdrawal processes.

Indeed, a system where a validator must take out its validator completely to recover all of its ETH and then join again is inefficient. This congests the exit queue with people just wanting to withdraw and penalizing those who really actually want to exit.

In addition, this represents a lot of risky operations for the validator which could cause him to lose his ETH in case of false manipulation when joining the network again.

A solution with two separate queues (one for exit and one for partial withdrawals) would simplify the system and reduce the risk. These queues would only be open to sleeping validators and therefore receiving neither work nor rewards. However, the implementation of a delay would be necessary in order to secure this process and prevent it from being used too much.

Conclusion

Here is a diagram recapitulating the overall system.

SQ SQ

SQ

Asleep Asleep

Asleep

SQ->Asleep SQ->Asleep

Awake Awake

Awake

Awake->SQ Awake->SQ

Awake->Asleep Awake->Asleep

WQ WQ

WQ

WQ->Awake WQ->Awake

Asleep->Awake Asleep->Awake

Asleep->WQ Asleep->WQ

PartialWithdraw PartialWithdraw

PartialWithdraw

Asleep->PartialWithdraw Asleep->PartialWithdraw

Exit Exit

Exit

Asleep->Exit Asleep->Exit

Asleep->Exit Asleep->Exit

PartialWithdraw->Asleep PartialWithdraw->Asleep

The black arrows represent the voluntary actions of the validators and the red arrows the actions forced by the protocol.

The green boxes represent active validators, orange inactive validators and gray pending validators.

This system has the advantage of allowing everyone to participate in securing the network.

It would even have the effect of reducing the differences due to luck between validators. A validator lucky enough to get a block with a lot of rewards (if there was a slashing event in the block or if tips are high) will then see his NRS

increase and will therefore have to compensate by spending more time asleep . This will give less fortunate validators the opportunity to increase their reward.

(or maybe this is a defect and only the rewards coming from the attestations should be taken into account for the NRS

and the rewards coming from the blocks must be excluded)

Another system would be where the number of validators is limited by a hard cap and a validator could only enter if another goes out. This combined with the inability to make partial withdrawals, would create a forced turnover. But I find this operation more risky because a validator could keep his place indefinitely and it would be complicated to exit to recover his rewards without knowing when he can enter again.

My proposal makes it possible to reduce the downtime of validators and thus improve the efficiency of the network. As mentioned at the beginning of this proposal, I am interested in your feedback on :

- The general idea of this proposal
- Feasibility and complexity of such a system
- Security, if it involves obvious security weaknesses that cannot be easily corrected