On the approaches for Keepers' selection in decentralized autonomous smart contracts execution networks

The PowerPool team

Winter 2023

Contents

1 Introduction

1.1 Keeper networks and their advantages

Almost every web3 protocol requires executing certain transactions, usually conditional transactions, triggered by on-chain events or time conditions. Ethereum and other EVM blockchains don't offer any method for these contingent actions to be done automatically at the appropriate times from inside any given smart contract. Thus, web3 protocols should be supplemented by external actors signing the transactions upon occurrence of the condition.

Keeper is an actor that executes particular smart contract functions according to some predefined condition. The multiple Keepers can coordinate under a specific operational logic or mechanism design and act as a decentralized network. The advantages of Keeper networks over standalone (centralized) bots are fault tolerance and robustness: for a big enough network, the probability of at least one keeper being available at the needed time is far more than that for any centralized bot.

However, the robustness and fail tolerance of decentralized Keepers networks significantly depend on the particular logic of keepers selection. This article provides an overview of available approaches for implementing probabilistic Keeper selection and our results for designing Keeper stake weighting functions.

Hereafter the following terms shall be used:

1. Task, also Job. A transaction or a series thereof which need to be conditionally executed by the network on behalf of a job owner.

2. Keeper. A member of the network that is assigned Tasks and carries them out.

1.2 Cost of no execution

When a certain task is delegated to a keeper or a network thereof, failure on their part may incur certain losses to the task owner, ensuring liability for which is important to promote stability of service. We shall define as the cost of no execution the lower bound of the direct loss suffered by the task owner when such a task fails to execute. For example, if the task is devoted to maintaining the target collateral ratio of a CDP position when a collaterisation ratio close to critical, the cost of no execution shall equal the liquidation loss. Hereafter we shall treat this as a value known or obtainable by a known function.

1.3 Centralised and decentralised keepers. Argument of robustness of a decentralised keeper network.

A centralized automation option is an appealing prospect since of its ease of launch and maintenance; rather complicated computations may be offloaded off-chain, and operation costs may thereby be reduced also. However, aside from the obvious downside of producing a trusted party (which means that failure on its part may lead to catastrophic losses, far in excess of those incurred by failure of any single keeper of a decentralised network), it does not provide as robust an entity as a large enough decentralised network. The same can hold even for large subsets of nominally decentralised keepers if they were to use the same RPC endpoint provider or hosting service or be in the same network segment, all of which provide a centralization point and face the failure simultaneously if the RPC will be down.

Indeed, the superiority of a decentralised configuration is easily shown: let there be a certain decentralised network of N keepers. We shall consider it

firstly to be a regular simple graph of degree $N-1$. There are two kinds of network failure we need to consider: total failure of a network and failure of such a set of nodes that the network breaks up into two connected components (deletion of the smallest separator). It is rather obvious that for practical N the total failure becomes practically impossible very quickly even if we assume poor uptime statistics (e.g. if each keeper has an uptime of 50%, then a network of 20 such keepers shall be entirely down with a probability of order $1e-6$, which already exceeds the robustness of a centralised network considered above). We now consider the question of the separator. If the graph be indeed regular with degree $N-1$, then trivially no separator may exist there. As a useful point we mention that the algebraic connectivity of such graph is N, which can be obtained straightforwardly by performing Gaussian elimination.

Now, for the sake of the argument, let the graph be irregular and let its smallest degree not differ considerably from N (i.e. be of lesser order). Then the Liu lower bound of vertex connectivity shall be of order $\sqrt{N}$, which is still a great number if N is great, and the probability of the entire separator (which may well be greater in cardinality than the lower bound, naturally) being excluded simultaneously will be insignificant by the same argument as before. A decentralised keeper network therefore exhibits superior robustness in addition to not introducing undesirable centralisation.

1.4 Cost of no execution as a metric by which automation may be governed

As mentioned above, the cost of no-execution is defined as the lower bound of the losses directly suffered by a task owner as a result of a failed task execution. Obviously, to prevent malicious actors from sabotaging Tasks, penalties of appropriate magnitude could be imposed on failed keepers to economically disincentivize possible failure. Since the cost of no-execution is one of the essential metrics applied to a task, it can be used as an input parameter for a probabilistic keeper selection method based on the keeper's skin in the game (a stake).

1.5 Variability of the methods whereby a keeper is chosen

A keeper may be chosen by a variety of methods, the particularities being dependent heavily on the nature of the job to be carried out. If a job is a routine task that does not jeopardise its owner greatly (i.e. has no cost of no-execution or, at the very least, a small one), the keeper may be chosen rather arbitrarily because the risks associated with keeper failure are largely negligible. Such cases merit employment of simple gas-efficient methods like the round-robin scheduling. On the other hand, tasks with significant costs of no-execution suggest weighting that gives some preferences to keepers that are either of good repute (where reputation may be constructed from time spent in the pool, cost-of-no-execution-weighted task count, etc.) or in possession of a reasonably large

(preferably of order of the cost of non-execution) stake which represents a slashable collateral. Furthermore, if a given task if of extreme importance and value, it may be split into subtasks as a further layer of security, the keepers of these subtasks being chosen according to some algorithm.

2 Keepers selection mechanisms

2.1 Algorithms of regular network scheduling in application to keeper networks

Hereafter a number of load-balancing algorithms are listed jointly with their potential analogues in keeper networks. Some of them are already used in existing keeper network solutions.

Load-balancing algorithm	Explanation	Analogue in on-chain network	Miscellanea
Round-robin	Sequential uniform distribution among the servers picked from a list	Sequential uniform selection of keepers from a list	Superior ease of implementation; full predictability of keeper selection
Round-robin with weighting	Every new process is assigned to a new server based on weighting	Every new task is assigned to a new keeper based on weighting	
Round-robin with priority	The processes are assigned according to a server priority ranking with lower-ranked servers acting as failure countermeasures	The tasks are assigned according to a keeper priority ranking (based on weight) with lower-ranked servers acting as failure countermeasures	Superior robustness and gas efficiency. Unless artificial incentives are introduced, the same keeper normally receives rewards

Least connection	The processes are distributed among servers in ascending order of connection number	The tasks are distributed among keepers in ascending order of task execution count	Though reward distribution is rather uniform and provides good keeper incentives, suboptimal keepers are likely to be selected. May be vulnerable to sybilline attacks since newly created keepers have the lowest possible task execution count.
Least connection, weighted	The processes are distributed among servers in ascending order of connection number and according to weights	The tasks are distributed among keepers in ascending order of task execution count with weights taken into account	
Least connection, locality-based	Requests from an IP address are sent to a selected group of servers based on locality, and the server with the lowest latency and load process request	Tasks are first assigned to a group of keepers, and then a keeper is chosen within that group by the least-connection method	
Least connection, locality-based with replication scheduling	Sender's IP address is bonded to a group of servers. Inside this group the less loaded server is chosen. If all servers are busy the available one is taken from another group (less loaded), and added to the new group [auto-re-grouping]	The sender of the task is bonded to a subgroup of Keepers. The "less loaded" Keeper will process the task. In the case that all keepers are busy there will be automatic re-group from the other group	

Least bandwidth	A server with the least bandwidth is chosen	The keeper that performs the lowest average number (over some horizon) of requests per block is chosen	
Destination Hash Scheduling	Server is chosen based on IP address of the request sender	Keeper is chosen based on the nature of the sender (which protocol or user asks for a job), particular network or other parameters	May be suitable for certain types of tasks; may provide insufficient robustness if keeper set be small
Source Hash Scheduling	Server is chosen based on IP address of request receiver	Keeper is chosen based on the nature of the Job	Same as above
Hash sticky	Requests from particular client are always processed by specific server	Particular job is always executed by a predefined Keeper	Provides inferior robustness

2.2 Gas auctions

The gas auction mechanism is exceedingly simple, far more so than most of the methods mentioned above, and has the benefit of practically ensuring the chosen keeper has the capacity of carrying out the transaction. Furthermore, no specific action on part of the keeper network is required because selection is automatic. The mechanism is based on the existing incentives the users employ to cajole miners into including their transaction in the next block: prospective keepers send transactions, and the one with the greatest gas provided is chosen, all other being reverted with errors at some subsequent point. The downsides of this simple method are evident: it imposes great losses onto keepers who have not been chosen because reverted transactions still cost gas.

An enhancement of this method, albeit at the cost of centralisation, is provided by the Flashbots pools. These pools provide costless reversion of failed transactions, alleviating the burden on the failed candidates and making the gas auction mechanism more viable. Naturally, an issue of culpability remains, because the task being carried out can only be checked post factum, because the task thereby is not guaranteed to be carried out at all, and because these properties make it difficult to impose corrective measures. Moreover, the keepers with the least ping to Flashbots will be chosen with a greater frequency. On a finite time-horizon, this would justify imposing a low-pass filter on the pings,

thereby showing that the actual keeper pool is less than the apparent one, and the entropy is correspondingly lower.

2.3 Uniform random selection

It is possible to randomly select a keeper from a set of keepers that have staked at least some minimum amount of tokens to partly or entirely cover the cost of no-execution in case of failure. This method is exceedingly simple, requiring only a single generation of a pseudorandom number and perhaps a composition thereof with some deterministic function (e.g. when a blockwise random number is employed, it is prudent to define a function that would ensure that the same keeper would not always be selected) and can be given robustness by adding onto it slashing rules of some description.

It has little downsides except that the size of the stake does not matter whatsoever (provided it exceeds the specified minimum). It is undesirable for reasons of possibly insufficient deterrence of malicious actors, since if a task has high costs of no-execution, it must either limit its execution to a very small pool of keepers. The latter problem is compounded by the fact that no stake weighting means one can produce a great number of accounts with minimum admissible stake and thereby increase his chances whilst decreasing the slashing penalties in absolute expression.

2.4 Nonlinear probabilistic selection

In order to enforce execution of tasks by keepers, it seems prudent to implement slashing of stakes of those who fail to do so when chosen, regardless of the reason of failure, with the aim of making malicious behaviour unprofitable. However, to prevent the penalty from tending to the lower bound of the admissible stake, it is also prudent to implement a weighting algorithm that would associate greater weights with keepers with greater stakes. The principal contribution of this article consists in examining different weighting functions by which it may be carried out. Once the weights are computed by means of some function, selection will be made trivial by computing the CDF of the resultant distribution and performing sampling based on it, nonlinearity being recovered from the shape of the CDF.

3 Generation of pseudorandom numbers on-chain

3.1 On the benefits of randomisation and sources thereof

Random, or rather pseudorandom numbers can be useful to a user of the blockchain in a variety of ways. The original whitepaper on Ethereum [4] suggested gambling, but a far more useful ability is that of choice between multiple candidate validators (as is the case for PoS Ethereum) or keepers [2] in order to introduce a degree of unpredictability in said choice and avoid unduly

skewing the benefit in favour of those who can provide largest stakes. This establishes the need to have randomness sources (hereafter randomness means pseudorandomness), and common blockchain principles obligate them to not rely on centralised off-chain data, such as that obtainable by calling an external random number generator.

An obvious possibility to do so would be to use the current block's hash as a seed for a conventional random number generating algorithm (e.g. a Mersenne twister), ensuring by the nature of the blocks that the number thus begotten shall be the same for everyone in the block and preventing cheating in this way. However, it opens the random number generator to manipulation by miners. Though it can be reasonably mitigated by introducing a delay of e.g. 6 blocks, such a vulnerability to collusion is still undesirable, for which reason wholly decentralised options merit exploration.

3.2 On the idea behind RanDAO

3.2.1 Naive algorithm

The idea consists of introducing randomness by taking in a large number of random numbers generated locally with arbitrary seeds by the willing participants, with a certain lower bound N_{lb} on the amount thereof, and combining them in some way to produce one number which will be taken as the output of the algorithm. This way is a deterministic (else a recursive need for random number generation is begotten) function of multiple arguments, usually corresponding to iterative XOR or an iterative composition of a hash function and summation. We shall use the iterative XOR hereafter. Let such function be $f(x_0, x_1, \cdots, x_n) : \mathbb{N}^n \to \mathbb{N}$. Let also the overbar denote exogenous variables. Then the simplest algorithm of this kind acquires the form of:

Algorithm 1 Simplest form of RanDAO

Require: $\overline{x}_0, \overline{x}_1, \cdots, \overline{x}_n; f : \mathbb{N}^n \to \mathbb{N}; n \geq N_{lb}; \overline{x}_i \in \mathbb{N}$
Ensure: $x \in \mathbb{N}$
$x_0, x_1, \cdots, x_n \leftarrow \overline{x}_0, \overline{x}_1, \cdots, \overline{x}_n$
$x \leftarrow f(x_0, x_1, \cdots, x_n)$

In order to properly analyse the suitability of such an algorithm, a number of demands is to be made of it, such as:

1. Fairness. By fairness we mean that the entropy distribution over the set of participants is not imbalanced.

2. Robustness. By robustness we mean that the algorithm must not exhibit sensitivity to the participants' malicious actions.

3. Unpredictability. This item requires no elucidation.

It is trivial to see that the algorithm proposed above violates the demand of fairness. Let us demonstrate this on the example of the function we elected to consider, that is, iteratively called XOR. XOR with a known number is a bijection. Moreover, it is an involutory function, which is trivially seen from its property of commutativity and the way identity and inverse are defined under it. From these considerations it stems that the sole actor responsible for the output of the DAO is he who submits the last number (which means that all entropy is concentrated on his submission), since he may reproduce the number generated thus far (as the current submissions and the function are public knowledge) and by means of a fast computation determine such an input that RanDAO's output would become c, where c is an arbitrary permissible number. Of course, picking the hash-sum composition as f would complicate such a task, but not make it outright impossible, since even though the last submitter would have to do what amounts to a brute-force search, he would still be the sole determinant of the output.

3.2.2 Commit-reveal algorithm

In order to disempower the last submitter it is necessary to decouple the processes of random number submission and public disclosure thereof. A natural way to do so is to require submitters to provide initially not the locally generated number itself, but rather its image under some fixed injective (or at the very least non-invertible) map and have them reveal the number itself only after all images have been collected. In such a framework, changing one's number would result to its image not coinciding with the one provided previously, which would be grounds to discard such a submission.
In practice, hash functions (which are not strictly injective) are employed to this end. With them, the algorithm becomes (here $g(x) : \mathbb{N} \to \mathbb{N}$ is the function defined above):

Algorithm 2 RanDAO with commit-reveal

Require: $\overline{x}_0, \overline{x}_1, \cdots, \overline{x}_n; g_0, g_1, \cdots, g_n; g : \mathbb{N} \to \mathbb{N}; f : \mathbb{N}^n \to \mathbb{N}; n \geq N_{lb}; \overline{x}_i \in \mathbb{N}$
Ensure: $x \in \mathbb{N}$

```
g_0, g_1, ..., g_n ← g_0, g_1, ..., g_n
Submission ends
x_0, x_1, ..., x_n ← x̄_0, x̄_1, ..., x̄_n
for j ← 0 to n do
    if g_j ≠ g(x_j) then
        Discard x_j
    end if
end for
x ← f(x_0, x_1, ..., x_m); m ≤ n
```

Now the latest submitter only has in his power 1 bit of information, same as everyone else (since he makes a binary choice of submitting his number or declining to reveal it, depending on which output is more favourable - he can still

compute both easily), but such a scheme would still be vulnerable to Sybilline attacks, since he who controls c submissions controls c bits of information. We shall first outline a method to further complicate such an attack by algorithmic design and then elucidate how RanDAO actually works.

3.2.3 Brief interlude on time-lock puzzles: towards VDF

In order to deprive the possible attacker of even the opportunity to make an informed choice regarding disclosure of his submission number, it would be sufficient to make the output of RanDAO not $f(\mathbf{x})$ itself, but rather $(V \circ f)(\mathbf{x})$, where $V : \mathbb{N} \to \mathbb{N}$ is a function which takes a great amount of time to compute. In such a case, the attacker would have to compute it at least twice, with the number of computations increasing exponentially with the amount of bits he wishes to exert informed control over. If the computation of this function requires more time than what is allotted for the number disclosure step, the attacker would not be capable of predicting the possible outputs (with some caveats discussed at a later point).
The astute reader would notice that this is not too dissimilar from how proof-of-work protocols function. That would be correct, except PoWs are closer in spirit to VDFs, which are discussed in the next section.
It is possible to devise such a time-lock puzzle with the same instruments that ensure blockchain cryptography: finite-order groups begotten by modular arithmetic. A classic choice is the RSA group begotten by arithmetic modulo a product of two primes that are not disclosed. An example of a time-lock puzzle would require one to, having been provided with an element y of that RSA group and a timing parameter t, compute y^{2^t} [6]. This requires one to either factorise the group order to employ the theorem of Fermat and Euler (which is scarcely possible) or perform t squarings in the group, which provides an execution time of t computational steps, if one disregards the inner workings of the squaring.
We could employ this time-lock puzzle in the manner described above and achieve some results with it, but it presents a problem: though it surely makes computing the output a long ordeal infeasible for the attacker to perform many times, it does not provide an easy way of checking the correctness of the output for a given input (not unless the group order is shared, which breaks the security of the algorithm irreparably) and therefore is prone to centralising power in the hands of the protocol.

3.2.4 Commit-reveal with VDF

In order to achieve paramount security we seek, we need to envision such a function $W : \mathbb{N} \to \mathbb{N}^2$ that, given an input, it returns an output and a proof such that the latter may be used to prove that output stems from input with little computational complexity (a good analogue for such a function is the elliptic curve signature generation algorithm). Here we outline one of the relatively recent developments in the field of practicable VDF functions, which is not meant to be an exhaustive presentation and rather an easy-to-follow illustration for

the reader.
Suppose there is, as in the time-lock puzzle before, the RSA group $G = (\mathbb{Z}/N\mathbb{Z})^\times/\{\pm 1\}$, where $N = pq$, with both of those numbers prime. This setup does have a secret key (the role of which plays the group order, and the public key counterpart to which is N), but it turns out that it is possible to construct a group of similar behavior without having any secret keyes. Such a method, however, relies on concepts far beyond the scope of the present article (such as ideal class groups and field extensions), and he who desires to know the details is referred to [1]. Regardless of which of the two methods is used to construct G, the algorithm (due to Wesolowski) will proceed equivalently [7]. Allow additionally there to be a hash function $H_G(x) = H(x) \ mod \ N$.
Allow y to be the input of the VDF. Then allow $g = H_g(y)$ to disrupt any structure associated with exponentiation (which is a homomorphism). The first part of the algorithm is then the time-lock puzzle identical to before (i.e. a problem of computing $z = g^{2^t}$ with a time factor t). The second part, however, is the ingenious verification algorithm.
Let P_{2k} be the set containing the first 2^{2k} primes. The verification party samples uniformly at random (this is done locally and so creates no recursive dependencies) $l \in P_{2k}$. The proving party then computes $\pi = g^{2^t/l}$, which may be done by any party irrespective of them holding a secret key in an amount of group operations with upper asymptotic estimate of $\frac{t}{log\ t}$, which is substantially faster than the t operations required in the forward direction. Having received π, the verification party accepts the proof iff $z, g, \pi \in G$ and $\pi^l g^{2^t \ mod \ l} = z$. It also turns out that the problem of finding a pair $\{z, \pi\}$ for a given y that satisfies the proof and is different from the honest one is equivalent to a root-finding problem, which is computationally infeasible, making the proof secure. With the described (or another) VDF, the algorithm becomes

Algorithm 3 RanDAO with commit-reveal and VDF

Require: $\overline{x}_0, \overline{x}_1, \cdots, \overline{x}_n; g_0, g_1, \cdots, g_n; g : \mathbb{N} \to \mathbb{N}; f : \mathbb{N}^n \to \mathbb{N}; n \geq N_{lb}; \overline{x}_i \in \mathbb{N}; W : \mathbb{N} \to \mathbb{N}^2$
Ensure: $x \in \mathbb{N}$
  $g_0, g_1, \cdots, g_n \leftarrow g_0, g_1, \cdots, g_n$
  Submission ends
  $x_0, x_1, \cdots, x_n \leftarrow \overline{x}_0, \overline{x}_1, \cdots, \overline{x}_n$
  for $j \leftarrow 0$ to n **do**
    if $g_j \neq g(x_j)$ **then**
      Discard x_j
    end if
  end for
  $x \leftarrow (W \circ f)(x_0, x_1, \cdots, x_m); m \leq n$

3.2.5 On the vulnerability of VDF to ASICs

Since we desire the VDF to be hard to compute for an attacker (preferably with some reserve), then the usage of ASICs presents a great problem. Suppose we desire to have a fivefold reserve of time. Suppose we know that the best ASIC tailored for the particular employed VDF is a mere five times faster than a regular computer. Then we would need the VDF to have a computation time twenty-five times greater than the duration of the submission reveal phase. This may quickly become infeasible, especially if we wish to, say, generate a random number each Ethereum epoch. The current solution sought for that problem by Ethereum Foundation is the proliferation of ASICs so that each member can be assumed to use one, but he who plans on using VDFs in his random number generators must be wary of the outlined issue.

3.3 Chainlink VRF

In general, VRFs employ operations on groups with no division quite like the algorithm outlined in the subsection on the VDFs, the principal benefit thus conferred being the ability to employ variations of the discrete logarithm-related Diffie-Hellman decisional assumption that underlies much of blockchain cryptography (since it holds for prime-order elliptic curves over finite fields). The decisional Diffie-Hellman assumption states that for certain groups (e.g. prime-order elliptic curves over the finite fields or subgroups of cyclic group of prime order for which the index is coprime with the group order) it is computationally impossible to distinguish between a composition of two random group elements and a third random independent group element (whence it trivially stems that the discrete logarithm is also hard in such groups).
In partiuclar, Chainlink uses a widely accepted VRF due to Goldberg *et al* [5]. Below an outline of this algorithm is produced; an explanation follows thereafter.

Algorithm 4 Goldberg VRF

Require: $\alpha \in \mathbb{Z}; G : \{g^k, k \in [0; q-1]\}; q$ prime$; G \leq E;$ **GCD**$([E : G], q) = 1; H_1 : \mathbb{Z} \to G/\{1\}, H_2 : E \to \mathbb{Z}_l; H_3 : \mathbb{Z} \to \mathbb{Z}_l; l$ is the desired length of the VRF output;
subscripts of integer numbers denote the admissible number of bits
absence of subscripts implies arbitrary length

Ensure: $PK \in G; \pi \in (G, \mathbb{Z}_l, \mathbb{Z}_l)$

Key generation

$x \leftarrow$ **RANDCHOICE**$[1; q-1]$

$PK \leftarrow g^x$

Production of a proof

$h \leftarrow H_1(\alpha)$

$\gamma \leftarrow h^x$

$k =$ **RANDCHOICE**$; k \in \{0; \ldots; q-1\}$

$c = H_3(g, h, g^x, h^x, g^k, h^k)$

$s = k - cx$ **mod** q

$\pi \leftarrow (\gamma, c, s)$

Verification

$u \leftarrow PK^c \cdot g^s$

$h \leftarrow H_1(\alpha)$

if $\gamma \in E$ **then**

$v \leftarrow \gamma^c \cdot h^s$

VALIDITY $\leftarrow c = H_3(g, h, PK, \gamma, u, v)$

end if

Production of output from proof

$\beta \leftarrow H_2(\gamma^{[E:G]})$

We now explain in plain words the operation of this algorithm. It has the following public parameters: the VRF input α, obtainable by any means from the consensus state and used as a seed; the cyclic group G of prime order in which the DDH assumption holds (it is assumed to be a subgroup of a group E of order indivisible by square of q, which imposes a requirement onto the index by the theorem of Lagrange. This group is needed as an auxiliary construction in case checking whether an element is in G is hard, and is defined by its property that such a check is easy for it); three hash functions that map an arbitrary-length input in bits to a group element, a group element to a number of specified length, and an arbitrary-length input to a specified-length input respectively. We first choose a secret VRF key x at random and generate a corresponding element of the cyclic group as a public key. In addition, a second generator for the same group is begotten as an element thereof corresponding to α by some rule. Observe that since h, g lie in the same cyclic group, there is a power w such that $h = g^w$. Then the DDH assumption reads: given a tuple (g^x, h), it is computationally infeasible to distinguish h^x from an independent point. Expanding the relation between the two elements gives the tuple (g^x, g^w) and a value $(g^w)^x = g^{wx}$, which is precisely the form DDH is stated in commonly; observe also that it means h^x is random for any computationally bounded observer. Then the principal part of the proof is produced by way of the Chaum-Pedersen algorithm as an output c of a hash function that takes the two generators and their x, k-th powers, where k is chosen at random with the caveat that it be no greater than the group order. In order to make a proof usable, however, we must endow the verifying side with the ability to compute such a function, i.e. to reconstruct the x, k-th powers without revealing x, k, so that the coincidence of values may be checked. We note that x-th power of g is known by the definition of the public key. Then a proof sent together with this key must comprise $\gamma = h^x$ and some way to reconstruct the $k - th$ powers. Once again, k itself must not be revealed so as not to undermine the security of the function. Instead, we use the group property to define $s = k - cx$ **mod** q, whence it follows that e.g. $(g^x)^c g^s = g^{cx} g^s = g^{cx+s} = g^k$. This definition reveals that it is sufficient for the verifier to know the public key and the triplet of values γ, c, s. To provide the ability to pass the VRF output implicitly, it is also defined as an image of γ raised to a known power under a known map.
In particular, Chainlink implements the VRF as suggested in the paper, i.e.:

1. Image of H_3 is the first l bits of SHA-256 output of its argument
2. H_2 is SHA-256
3. H_1 is a public-key-dependent algorithm so that a fresh random oracle is created for each public key, and brute-force attacks are made substantially harder. In particular, the algorithm proposed in the paper suggests a technique that is outlined below.

Algorithm 5 Goldberg H_1

Require: $\alpha; PK; H : \mathbb{Z} \to \mathbb{Z}; EC : \{y^2 = x^3 + ax + b; \|EC\| = qf\}$
Ensure: $\gamma \in (\mathbb{Z}, \mathbb{Z})$
  $i \leftarrow 0$
  $h \leftarrow H(PK, \alpha, i)$
  if $\exists p \in EC : p(0) = h$ **then**
    $\gamma \leftarrow (h, (h^3 + ah + b)^{0.5})^f$
  end if
  $i \leftarrow i + 1$

Chainlink implementation provides an additional degree of separation between the consumer of the random value and the generator: when a contract requests a random value, it is generated, and the proof is sent to the VRF contract, which verifies it before passing it to the consumer, thereby relieving him of the burden of verification.

3.4 Random number generation for chains that possess an inbuilt random number generator

Certain chains, e.g. PoS Ethereum, possess an inbuilt mechanism of generation of pseudorandom numbers for purposes of selecting validators of blocks. For instance, Ethereum blocks have a *randao-reveal*, formerly *diff*, field in it, meant to act as a tool of choosing the validator. Since block state is accessible by smart contracts, it can be used as a pregenerated on-chain pseudorandom number. However, if this number is used raw, then the keeper thereby chosen will be the same over the course of the entire block, which may be undesirable. For this reason, some operation may be used to combine it with a dynamically changing variable in order to produce a variable output. For instance, one may take the index of the last registered job, add it to said pregenerated number and obtain a different number. Moreover, it may then be possible to cheaply choose the keeper for task execution by taking the remainder of the number obtained above modulo the cardinality of the keeper set. When a chain lacks a random number generator, though, it must be constructed artificially. We shall provide the example of doing so by outlining the means by which RANDAO works on Ethereum.

3.5 RANDAO of Ethereum

VDFs are not feasible for Ethereum with its fast block updates, not unless random numbers are rarely regenerated (they are not, at least because proof-of-stake requires RNG to choose a block proposer, and the current Beacon Chain regenerates the random number every epoch). The actual mechanism by which RanDAO functions is as follows:

1. In an allotted time frame (e.g. six blocks' time) every participant sends

to the RanDAO contract some ETH as pledge and the Keccak256 hash of a number s he generates

2. In an allotted time frame, everyone who submitted a hash must send a second submission with the number s. The number is checked against its hash and stored if it passes.

3. The random number is resolved with a preset function f, successful pledges are refunded, and the number is sent to the contracts that requested it. A bonus consisting of fees paid by other contracts that consume the random number and (optionally) the confiscated unsuccessful pledges is divided between the participants also.

The following failure conditions are imposed:

1. If multiple hashes are submitted in the first phase, only the first is kept.

2. If the participant submits the hash but does not reveal the number, his pledge is confiscated just as if he revealed a wrong number

3. The RNG fails if one or more s instances are not revealed or if the amount of hashes gathered is insufficient (i.e. lower than some parametric bound). Confiscated ETHs is divided equally and send to other participants who revealed s at the second phase. The fees paid by other contracts are refunded.

4 Keeper weighting functions

4.1 Criteria of optimality

Assume there is a countable set of keepers S. Assume that there is a map $p : S \to \mathbb{R}_{0+}$ that has as its image the stake of a particular keeper. Assume furthermore that there is a **candidate subset** consisting of all keepers that satisfy a particular condition imposed by a contract S_c. It is desired then to suggest and study such probability measures on the event space of a single keeper being chosen, with the following economically motivated criteria:

1. The function ought to be gas-optimal, i.e. expressible primarily algebraically

2. The measure ought not to unduly skew the probability in favour of the big stakers in order to prevent low-staking keepers from being virtually never chosen despite meeting the requirement imposed

3. The measure ought to nonetheless provide benefit to him who stakes more

4. The measure ought to have a parametric saturation input which can be understood to encode the cost of no execution of the contract considering this candidate set

4.2 Secondary criteria for functions

We posit the following necessary and tentative consequences of the criteria stated above:

1. The measure is a function that is Lipschitz on $\mathbb{R}_{0+}$
2. The measure ought to be monotonous. This criterion is not fixed, and functions that do not enjoy this property shall also enter consideration
3. The measure ought to have a finite limit at infinity. If we define limit convergence speed as the inverse of a variation of input over which the function diminishes by a factor of e, it must not be too small. In other words, the asymptotic behavior must not be bounded by a slowly varying function.
4. Assuming derivatives exist everywhere, it is curious to see whether monotonous behavior is a desirable property thereof.
5. If the function be saturating, then the derivative thereof obviously must be monotonously decreasing and tending to zero at infinity.

We operate in a candidate subset of a fixed size N, wherein all stakes are expressed as shares of the sum of the stakes over the entire subset in order to ensure independence of absolute values. Then the domain of p shifts from $\mathbb{R}_{0+}$ to $(0; 1)$, and all properties posited here and afterwards are understood to hold on it. We allow the function $x(j)$ to map the candidate number to his stake size.

4.3 Functions formulated from general considerations

4.3.1 Linear

$$p(x) = x$$

As seen from the plot, it heavily favours the whales. Since it has an especially simple form, we can already say that the probability of being chosen is equal to the share of the total stake, and if someone has a dominating share, he would be greatly favoured. The function is Lipschitz and monotonous, but its derivative is constant, which leads to the whales being favoured (i.e. no diminishing returns - the influence per stake unit is constant regardless of the stake already put in).

4.3.2 Square root

$$p(x) = \sqrt{x}$$

The derivative here is $\frac{1}{2\sqrt{x}}$. This is an admissible function since it is Lipschitz, and its derivative is monotonously decreasing (i.e. an effect of diminishing returns is extant). However, it has no saturation possibility.

4.3.3 k-th root

$$p(x) = x^{1/k}$$

The derivative here is $\frac{1}{kx^{(k-1)/k}}$. Alike the previous one.

4.3.4 Logarithm

$$p(x) = log(x)$$

The derivative here is $\frac{1}{x}$. This function, it must be noted, requires shifting on the domain we have chosen, but this is not a great problem. It shares much similarity with the square root, except that its second derivative ($\frac{-1}{x^2}$ versus $\frac{-1}{4x\sqrt{x}}$ for square root) is a faster-diminishing function, and as such the diminishing returns have a more stringent penalty.

4.3.5 Hyperbolic tangent

$$p(x) = \tanh(x)$$

The derivative here is $sech(x)^2$. The function and the derivative have satisfactory properties, but are obviously not algebraic and therefore gas-inefficient. We shall hereafter consider functions of similar behavior and algebraic nature.

4.4 Functions formulated by imposing demands on the derivative

The functions considered above are either non-saturaing or non-algebraic, both of which are undesirable properties to have. We now consider functions constructed as antiderivatives of specially chosen functions. We firstly demand of the derivatives the following properties, relaxation of which may yield other special cases:

1. Limit of zero at infinity
2. Nonnegativity on $\mathbb{R}_+$
3. Monotonously decreasing behavior

4.4.1 Sigmoidal functions with an algebraic derivative

A natural suggestion is to use functions of the kind $\frac{1}{x}$. However, they require regularisation so that their behavior is monotonous, and no singularities manifest in the domain. This may be done with a shift: $\frac{1}{x+s}$ and generalised to govern the speed of decrement to $\frac{1}{x^n+s}$. However, such functions do not have algebraic antiderivatives. Notably, for $n = 2; s = 1$ the antiderivative becomes an arctangent. We do not pursue this general class further because they lack a way to systematically define the antiderivative of a general function, because atan computation is unnecessarily complicated in Solidity, and because a simple modification may be thought of to remedy both issues.

4.4.2 Algebraic sigmoidal functions

Modifying the above proposal in a simple way, we may recover a family of algebraic functions with algebraic antiderivatives. Let the derivative be of the form $\frac{1}{(1+(x+s)^n)^{(n+1)/n}}, n > 1, n \in \mathbb{N}$. The antiderivatives are then of the form $\frac{x+s}{(1+(x+s)^n)^{1/n}}$. Such functions manifestly satisfy all demands made of them; furthermore, the location of the saturation and the speed thereof may be governed by two parameters: if we generalise the function to read $\frac{ax+as}{(1+(ax+as)^n)^{1/n}}$, the parameters are the scaling factor a and the shift s. By means of these one may make the saturation manifest at a desired value. However, doing so imposes a demand that n be even to prevent shifting to a section of undesirable behavior. Moreover, this allows to introduce a tail on the lower end of the admissible argument range where the function is both negligible and slowly growing, imposing an effective soft lower limit on the stake (so that it is virtually useless to stake less than that). This property may even be used to circumvent the need to have a logical expression demanding a hard lower stake.

4.4.3 Soft-cap algebraic sigmoidal functions

One may desire to artificially cap the stake size. It is possible to do so via contracts, but another simple realisation is a soft cap that would work by way of market mechanisms. To devise such a cap, we need a function similar to those of the AS family, but admitting negative derivatives and tending to a low value or even zero. By happenstance we have discovered that functions of the kind $\frac{x}{(1+x^n)^{1/k}}, n > k$ satisfy such a demand on $\mathbb{R}_+$ and are still algebraic. Such functions can also be extended to admit scaling and shifts of the argument, just like the ones in the previous subsubsection.

4.4.4 Maclaurin series approximation for the ASF2

We now consider the Maclaurin series for the ASF2. It is known that such approximations hold only locally, but, since we rescale the interval to unity anyhow, they can be very useful to save gas. Recall that the estimate for the remainder of a Taylor series about a is

$$Q\frac{(x-a)^n}{n!}$$

Where Q is the upper bound on the n - th derivative of f, and n is the first omitted power.
Terms of the Maclaurin approxiamtion of the ASF2:

0. $f(0) = 0$
1. $f'(0) = (x^2+1)^{-3/2}\Big|_0 = 1$
2. $f''(0) = \frac{-3x}{(x^2+1)^{5/2}}\Big|_0 = 0$

3. $f'''(0) = \frac{12x^2-3}{(x^2+1)^{7/2}}\Big|_0 = -3$
4. $f''''(0) = 0$
5. $f'''''(0) = 45$

The upper bounds of derivatives grow very quickly, and it becomes apparent that this choice is poor.

4.4.5 Maclaurin series approximation for the tanh

Terms of the Maclaurin approxiamtion of tanh (note that the upper bounds of the derivatives grow slower than for ASF2):

0. $f(0) = 0$
1. $f'(0) = 1$
2. $f''(0) = 0$
3. $f'''(0) = -2$
4. $f''''(0) = 0$
5. $f'''''(0) = 16$

Suppose we pick terms up to f'''. Then we have the following approximant:

$$\tilde{f}(x) = x - \frac{1}{3}x^3$$

This is a poor approximant, and going higher raises the upper bound of the derivative, producing high estimates of error. We must conclude that it is a poor idea to use Maclaurin approximation.

4.4.6 Lagrange-Chebyshev approximants

Suppose we pick the order of the approximant n. Then we can pick n Chebyshev nodes (incidentally, we are working with $[0; 1]$ - (almost) the natural domain of the Chebyshev polynomia), corresponding to points equidistributed on a circle with $[-1; 1]$ being its diameter, and build a Lagrange approximant on them. The reader desiring a closer look into Chebyshev interpolation ought to consult any book on numerical analysis, for example [3].
In order to transform $x \in [0; 1]$ into $z \in [-1; 1]$, define $z = 2x - 1$. Pick n Chebyshev nodes, evaluate $f(x)$ on these nodes, and build a Lagrange approximant from these values. Refer to the code I provide to get a demonstration of the good properties of the polynomial thereby obtained. It must be noted that the gas cost of the polynomial I give as an example (which is of order 5) will be about $5 * (6 + \Sigma_1^5 i) + 6 * 3 = 123$, which is about twice better than ASF2. Note, however, that the parameter update will become a two-step process necessitating high gas cost on generation of new ground truth values and Lagrange

interpolation thereof. This may be mitigated if the job owner is disallowed to set his own maximal stake and instead has to accept that it will be the result of reflecting minimal stake about the saturation point, in which case the coordinate transform at parameter update is linear and can have no inflection points. Gas cost estimates are everywhere obtained with elementary costs taken from [8].

4.5 Characteristic parameters for algebraic sigmoidal functions

Let λ, τ be the characteristic parameters. We define λ as the normalised value of the cost of no execution, and τ as the normalised threshold demanded for influence per token staked to start growing appreciably. Consider a function of the AS family: $y = \frac{a(x+s)}{(1+(a(x+s))^n)^{1/n}}$. Then λ, τ with tolerance ϵ are such values of x that $y'(x) - \epsilon = 0$. $y'(x) = a((ax+as)^n + 1)^{-\frac{1}{n}-1}$

$$a((ax+as)^n + 1)^{-\frac{1}{n}-1} = \epsilon$$

$$(ax+as)^n + 1 = (\frac{\epsilon}{a})^{-\frac{n}{n+1}}$$

$$a(x+s) = ((\frac{\epsilon}{a})^{-\frac{n}{n+1}} - 1)^{\frac{1}{n}}$$

$$x = \frac{((\frac{\epsilon}{a})^{-\frac{n}{n+1}} - 1)^{\frac{1}{n}}}{a} - s$$

Evaluate at λ, τ:

$$\lambda = \frac{((\frac{\epsilon}{a})^{-\frac{n}{n+1}} - 1)^{\frac{1}{n}}}{a} - s$$

$$\tau = -\frac{((\frac{\epsilon}{a})^{-\frac{n}{n+1}} - 1)^{\frac{1}{n}}}{a} - s$$

Then

$$\lambda - \tau = 2\frac{((\frac{\epsilon}{a})^{-\frac{n}{n+1}} - 1)^{\frac{1}{n}}}{a}$$

$$\frac{\lambda - \tau}{2} = \frac{((\frac{\epsilon}{a})^{-\frac{n}{n+1}} - 1)^{\frac{1}{n}}}{a}$$

$$\frac{(\lambda - \tau)^n}{2^n} = \frac{(\frac{\epsilon}{a})^{-\frac{n}{n+1}} - 1}{a^n}$$

From here a is recovered, and substitution recovers s.

4.6 Certain expressions for evaluation of the functions and use thereof in selection of random weighted keepers

Since choice of a keeper according to his weight may be done by sampling a random variable in the range $[0; 1)$ and choosing the index of the first greater value of the CDF as the index of the keeper chosen, we shall need to formulate the CDF for a given expression. We also formulate for the sake of completeness and ease of understanding the PMF, the expected amount of selections before a keeper is chosen, and the expected payoff per round. All functions are given as those of the keeper index j. Let the contract reward be constant at r. Let the losses incurred each round be constant at l. Stake of a keeper is given by the function $x(j)$. Formulae are given without shifts and scaling of the variable x because those operations may be included via a composition.

4.6.1 Linear function

PMF $P(j) = x_j$
CDF $F(j) = \Sigma_{k \leq j} x_k$
Expected time till chosen $T(j) = \frac{1}{P(j)} = \frac{1}{x_j}$
Expectation of payoff $V(j) = r x_j - l$

4.6.2 Square root function

PMF $P(j) = \frac{\sqrt{x}_j}{\Sigma_j \sqrt{x}_j}$
CDF $F(j) = \Sigma_{k \leq j} \frac{\sqrt{x}_j}{\Sigma_j \sqrt{x}_j} = \frac{\Sigma_{k \leq j} \sqrt{x}_k}{\Sigma_j \sqrt{x}_j}$
Expected time till chosen $T(j) = \frac{1}{P(j)} = \frac{\Sigma_j \sqrt{x}_j}{\sqrt{x}_j}$
Expectation of payoff $V(j) = r \frac{\sqrt{x}_j}{\Sigma_j \sqrt{x}_j} - l$

4.6.3 Logarithm

PMF $P(j) = \frac{\log x_j}{\Sigma_j \log x_j} = \frac{\log x_j}{\log \Pi_j x_j}$
CDF $F(j) = \Sigma_{k \leq j} \frac{\log x_j}{\Sigma_j \log x_j} = \frac{\Sigma_{k \leq j} \log x_k}{\Sigma_j \log x_j} = \frac{\log \Pi_{k \leq j} x_k}{\log \Pi_j x_j}$
Expected time till chosen $T(j) = \frac{1}{P(j)} = \frac{\log \Pi_j x_j}{\log x_j}$
Expectation of payoff $V(j) = r \frac{\log x_j}{\log \Pi_j x_j} - l$

4.6.4 ASF(n)

PMF $P(j) = \frac{\frac{x_j}{(1+x_j^n)^{1/n}}}{\Sigma_j \frac{x_j}{(1+x_j^n)^{1/n}}} = \frac{x_j}{(1+x_j^n)^{1/n} \Sigma_k \frac{x_k}{(1+x_k^n)^{1/n}}}$
CDF $F(j) = \Sigma_{m \leq j} \frac{x_m}{(1+x_m^n)^{1/n} \Sigma_k \frac{x_k}{(1+x_k^n)^{1/n}}}$

Expected time till chosen $T(j) = \frac{1}{P(j)} = \frac{(1+x_j^n)^{1/n}\Sigma_k \frac{x_k}{(1+x_k^n)^{1/n}}}{x_j}$
Expectation of payoff $V(j) = r\frac{x_j}{(1+x_j^n)^{1/n}\Sigma_k \frac{x_k}{(1+x_k^n)^{1/n}}} - l$

4.6.5 SCASF(n,k)

PMF $P(j) = \frac{\frac{x_j}{(1+x_j^n)^{1/k}}}{\Sigma_m \frac{x_m}{(1+x_m^n)^{1/k}}} = \frac{x_j}{(1+x_j^n)^{1/k}\Sigma_m \frac{x_m}{(1+x_m^n)^{1/k}}}$
CDF $F(j) = \Sigma_{v \leq j} \frac{x_v}{(1+x_v^n)^{1/k}\Sigma_m \frac{x_m}{(1+x_m^n)^{1/k}}}$

Expected time till chosen $T(j) = \frac{1}{P(j)} = \frac{(1+x_j^n)^{1/k}\Sigma_m \frac{x_m}{(1+x_m^n)^{1/k}}}{x_j}$
Expectation of payoff $V(j) = r\frac{x_j}{(1+x_j^n)^{1/k}\Sigma_m \frac{x_m}{(1+x_m^n)^{1/k}}} - l$

5 Gas cost estimation

As we have stated, the cost of the algebraic sigmoid when $n = 2$ is not much greater than that of the square root. Indeed, observe the form of the function, where a, s are fixed

$$y = \frac{a(x+s)}{(1+(a(x+s))^n)^{1/n}}$$

If n is 2:

$$y = \frac{a(x+s)}{\sqrt{(1+(a(x+s))^2)}}$$

It is readily observed that the gas cost comes from performing one addition and one multiplication, storing the value of $a(x+s)$ thus obtained in memory, performing one further multiplication and one addition to obtain the square of the denominator, taking the square root, and dividing the numerator by the denominator. Therefore, the overhead in comparison to simple square root is about 21 gas (with prices taken from Ethereum's yellow paper). It is sensible to ask what is the cost of the square root computation. Commonly the square roots are computed with seven iterations of Newton-Raphson with up to 14 additional bitshifts to conditionally transform the root guess and the value the root thereof is desired (also commonly called the Babylonian method). Bitshifts cost 3 gas as per EIP-145, IF statements amount to a JUMPDEST and a JUMP, costing 9 gas in total, and the inner comparison costs 3 gas. Newton-Raphson iterations consist of a bitshift, an addition, and a division. Since the lower root estimate is taken, a division, a comparison, and a conditional are compounded. This gives the aggregate upper bound of cost per such operation as $14 * 3 + 7 * 12 + 7 * (3 + 3 + 5) + 5 + 3 + 9 = 220$ gas. Therefore, with the overhead, the algebraic sigmoid will have the approximate gas cost of $220 + 21 = 241$ per evaluation operation performed, or $220 + 29 = 249$ if $a(x+s)$ is not cached.

6 Plots

Since the ASF functional family is the most promising one overall in terms of cheapness of evaluation and possession of desirable functional and differential properties, hereafter we give plots of weights, PMF, CDF, and the expectations of roudns until a given keeper is chosen and his reward given a number of prior keeper stake distributions.

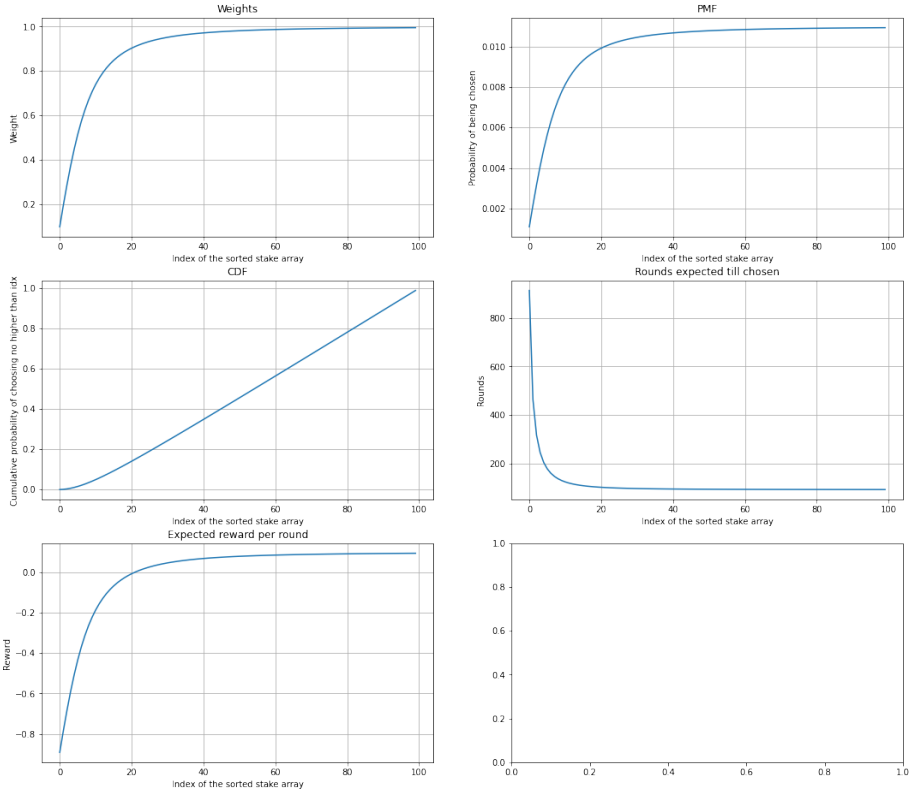

Figure 1: Plots with uniform stake prior

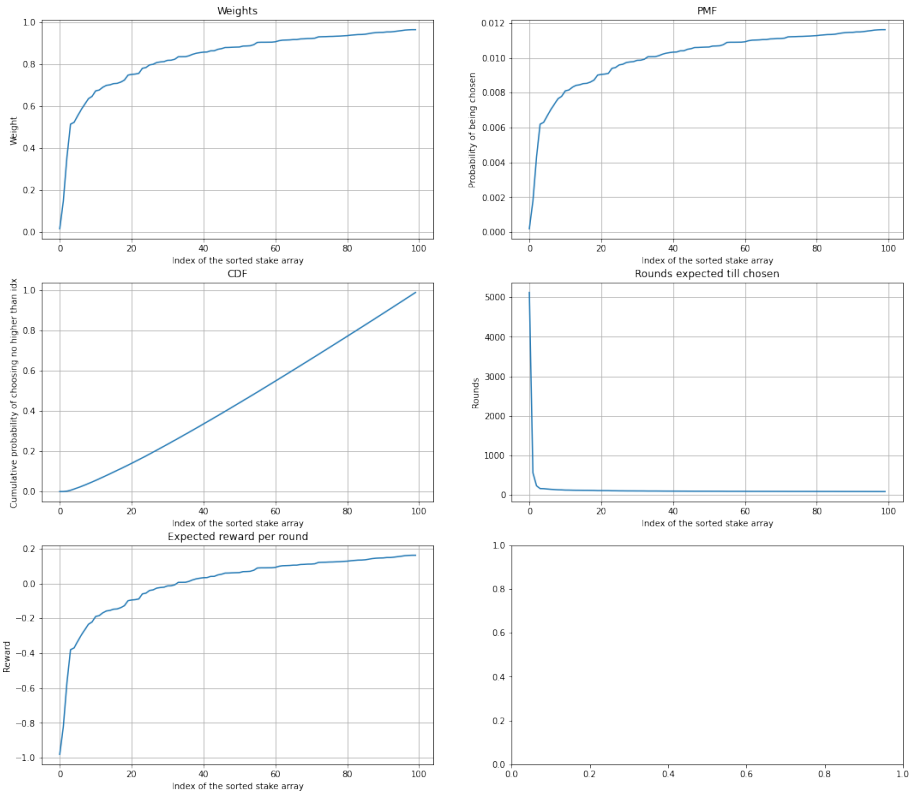

Figure 2: Plots with normal stake prior

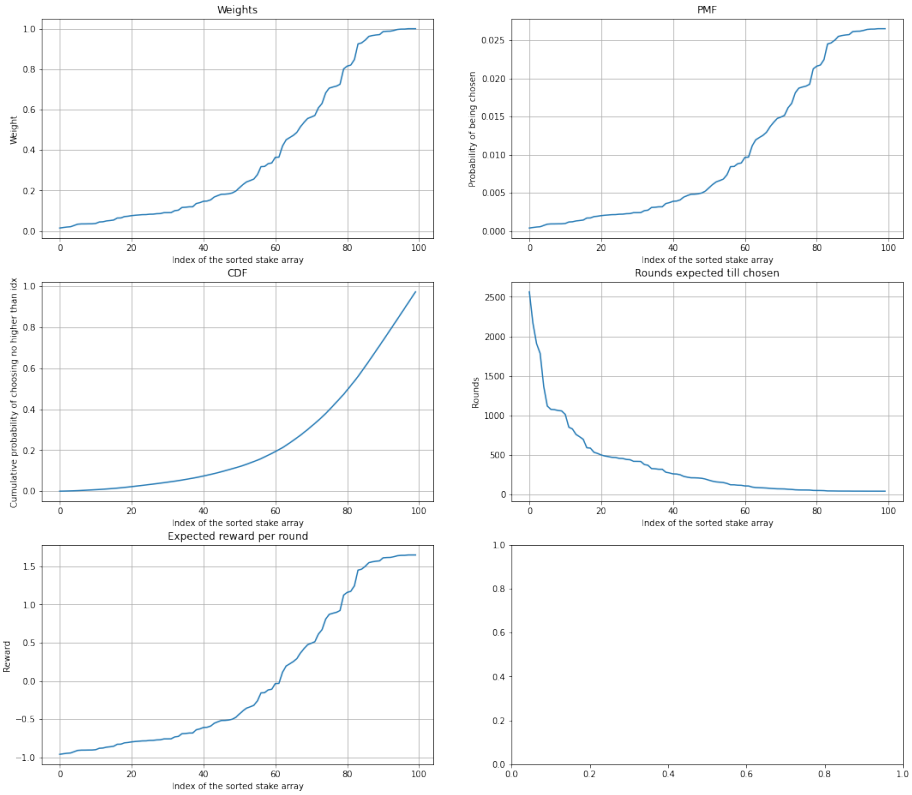

Figure 3: Plots with heavy-tailed (Levy) stake prior

Figure 1 corresponds to a prior uniform stake distribution of keepers; figure 2 gives the functions for normal distribution, and figure 3 models a possible real-world situation where whales are quite possible by means of a heavy-tailed (and thus outlier-admitting) Levy distribution.

7 Further research

First, our results for selecting keepers can also be applied to a general decentralized service provisioning network. For example, the combination of random selection with a stake weighting optimized for some critical system parameter (such as a cost of no-execution in the case of keepers) proposed in this article could be used to efficiently assign tasks to executors depending on their stakes, thus providing improved network reliability.
Secondly, a number of additional features may be implemented for the purposes of enhancing the keeper experience. We present them in the form of a list

1. Timeouts for keepers. The rationale behind it is that if a keeper with great stake is timed out, he cannot be chosen, and such a measure has an effect of making the reward distribution more uniform, which is an overall desirable outcome. Furthermore, slashed keepers can be by the same measure timed out for a considerable time, making attacks even harder.

2. It is entirely within the job owner's power to set the expected no-execution cost, which often cannot be verified by the pool manager's contract, to a level far above the actual value, thereby obtaining both cheap execution and great security. Furthermore, if he is reimbursed by the reduced keeper stake, Task owner has the opportunity to drain the network of value by posting cheap jobs with excessive estimated non-execution costs. Obviously, economic mechanisms should be proposed that balance the defined cost of no-execution with the size of the payoff for successful execution. Thus, one of the future work directions is to provide an algorithmic pricing rule bonding the cost of no-execution, execution fee, and the keepers' stake.

3. Finally, a further incentive to stay in the pool can be provided to keepers in the form of income sharing, whereby a portion of the reward for completing a job or a slashing event is allocated to all keepers who have not been chosen for completion, perhaps except for recent offenders (which is a parametrically defined category). A promising way to do so is to weight the portion of the shared reward by a score that takes into account the time a certain keeper has not been chosen in a row relative to what his stake would suggest. Naturally, this entails keeping auxiliary information (and maybe even a list of choice event time expectations so that appropriate aggregation may be carried out), but such a mode both prevents effortless farming of rewards with no useful work being carried out and lowers the

keeper's risk because even though his tokens may undergo depreciation without value generation (as is the case when they are staked, and their holder is not chosen for a long time), a part of this loss would be covered by the reward share. Moreover, he is thereby also incentivised to stay in the pool, since if he fails to be chosen again, his share of the proposed allowance would grow, and if he is chosen, then he receives a substantial reward.

8 Conclusion

In this article, an overview of known keeper selection algorithms and sources of on-chain randomness generation was presented. In order to design a keeper selection approach that could ensure the robust operation of a decentralized and permissionless keeper network, the cost of non-execution was defined as the key parameter on which to base stake-weighted random keeper selection. Based on the requirements for a function that defines the probability of keeper assignment for a task based on the cost of no-execution and the keeper's stake, the family of algebraic sigmoidal functions was selected, and a pre-processing transform of scaling maximal stake value to unity was chosen.

The main properties of such functions are positive definiteness and saturation at a given finite value, where the entire positive real half-axis is divided into three regions: (1) the region of extremely small stakes, which corresponds to the left tail of the sigmoidal function and provides no significant weight (and probability of being selected) per stake, (2) the region of the stake of comparable magnitude to the cost of no execution, which is used as the saturation location parameter, (3) and the region of excessive stake, which corresponds to the right tail of the sigmoidal function (i.e. the post-saturation region) and provides a rapidly diminishing increase of weight per token staked, greatly limiting the dominance of huge stakes. It is important to note that the function can also be reconfigured to provide a soft cap on the stake size.

For the given family of functions, gas costs were calculated from theoretical data (elementary operation cost as per the Ethereum Yellow Paper), and the cheapest function in terms of gas was selected. The possible cheapening of gas cost for using the function was proposed via approximating it by a polynomial on pre-calculated Chebyshev nodes, and it was found that it is possible only if the maximum stake for purposes of payment ceases to be an independent parameter (i.e. if the coordinate transform for configuring the saturation parameter is monotonic).

9 Bibliography

References

[1] Ingrid Biehl et al. "A signature scheme based on the intractability of computing roots". In: *Designs, Codes and Cryptography* 25.3 (2002), pp. 223–236.

[2] Lorenz Breidenbach et al. "2.0: Next steps in the evolution of decentralized oracle networks". In: *White paper, Chain-Link* (2021).

[3] R.L. Burden and J.D. Faires. *Numerical Analysis*. Cengage Learning, 2010. ISBN: 9780538733519. URL: https://books.google.ru/books?id=zXnSxY9G2JgC.

[4] V Buterin. *Ethereum whitepaper: A next-generation smart contract and decentralized application platform [White Paper]*. 2013.

[5] Dimitrios Papadopoulos et al. *Making NSEC5 Practical for DNSSEC*. Cryptology ePrint Archive, Paper 2017/099. https://eprint.iacr.org/2017/099. 2017. URL: https://eprint.iacr.org/2017/099.

[6] Ronald L Rivest, Adi Shamir, and David A Wagner. "Time-lock puzzles and timed-release crypto". In: (1996).

[7] Benjamin Wesolowski. *Efficient verifiable delay functions*. Cryptology ePrint Archive, Paper 2018/623. https://eprint.iacr.org/2018/623. 2018. URL: https://eprint.iacr.org/2018/623.

[8] Gavin Wood et al. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* 151.2014 (2014), pp. 1–32.