Phala Network Demand-End Tokenomic Design

Samuel Häfner
samuelhaefner.github.io
samuel.haefner@gmail.com

December 22, 2021

This proposal discusses a pricing mechanism for the Phala network, as requested in RFP #2. The RFP asks for a pricing model on the demand side; i.e., to sketch how PHA could be employed to allocate scarce computing resources among the smart contract users.[1]

1 The General Idea

The demand side of Phala can be broadly described as follows. There are individual *users* who want to employ *apps* to run certain computations. An app consists of a multitude of related smart contracts together with a front-end. Apps are deployed by *developers* who might themselves want to make use of them.

The general idea to allocate the network's computing power is to use an *app-centered approach with two layers*. The first layer allocates the network's computing resources among the different apps. On the second layer, the apps distribute their computing power among their respective users.

Specifically, users are allowed to stake PHA into the apps of their choice. The network then allocates the computing resources among the apps according to their shares of the total amount of staked PHAs.For the second layer, this proposal suggests different mechanisms that could be used by the apps. The apps are free to employ these mechanisms or to develop their own solutions. In any case, the mechanism on the second layer should ensure that first-layer stakers earn some revenue from their investment.

1.1 First Layer: Allocating Resources among the Apps

The number of resources allocated to the app depends on how much PHA the developers stake. In particular, if the total computing power in the network is P, there are n different apps, and app $i \in \{1, ..., n\}$ stakes an amount s_i, then the amount P_i of computing power reserved for app i is[2]

$$P_i = \frac{s_i}{\sum_{j=1}^{n} s_j} P.$$

Staking happens through a crowdfunding module to which all PHA holders can contribute. The module might have a cap on the total stake, which the developer can dynamically adjust. An app can remain funded forever, but users can take out their stakes at any time (possibly subject to an unbonding period). This ensures that once users stop wanting to use the app or deem the app as not staking-worthy, then there won't be any resources reserved for the app anymore, either.

Apps will also be given the possibility to issue their own *designated app token (ATO)*. In that case, the crowdfunding module issues ATOs to the contributors at a one-to-one PHA–ATO ratio. At any point in time, the staked PHA can be redeemed by transferring back the same amount of ATO. ATOs are exclusive to an app. They can be used to consume an app's computing

[1] The supply of computing resources is remunerated through a miner subsidy mechanism, which is already established and well described. The basic idea is to steadily remunerate miners for being in the network, where miners can choose payouts at any time. Possible payouts depend on a miner-specific value promise that grows over time according to a pre-defined formula, which takes computing power into account. Miners have to stake PHA when registering.

[2] Technically, the computing power that an app receives will be converted into a specific budget of CPUs that an app can claim. The apps can then choose which miners to use (potentially in order of their stakes), upon which their smart contracts are loaded in the respective secure enclaves.

resources and possibly allow participating in an app's governance. I envision that larger apps might want to have their own ATO both for community building and to decentralize decision-making.

1.2 Second Layer: Allocating an App's Resources to its Users

On the second layer, we need to think about how the computing power of a given app is allocated among its users. The basic idea is that the apps either use PHA or — if they have them issued — the designated app tokens (ATOs) to do so.

The allocation can happen in at least four different ways (but mainly we should leave it open to the creativity of the apps' teams):

1. *Based on 1st-Layer Staking*: This variant allocates the computing resources of an app proportional to the stake put into the app's staking module. That is every staker obtains a share of the app's resources that is equal to his or her share in the total stake of the app. There is an open question of whether and how we could allow users to trade the computing resources that they have in an app on an open market.
2. *Based on 2nd-Layer Staking*: This variant allows users to obtain a share of the app's resources by staking in a second-layer staking module additional PHA or the ATO that they obtained or bought on the market. As for the first layer described above, the share of the app's computing resources that a user obtains is proportional to his share in the total stake by all users.
3. *Transaction Fees*: The app might require users to pay a fee per transaction. There are at least two, standard ways of determining the fees:
   (a) *App-Chosen Fee*: Here, the app determines a fee that is proportional to the computing complexity of a smart contract call. The fee might also depend on the current usage of the app. The particular curve is set by the developer or, if existent, by the app's governance.
   (b) *User-Chosen Fee*: Here, I envision that users can set a fee and that the app then runs the submitted transactions according to their fees. That is, transactions with a higher attached fee are processed before transactions with a lower fee.

   A combination of the two fee-setting mechanisms is also possible, similar to Ethereum after the London upgrade: The total fee consists of a base fee set by the app and an additional tip that affects how fast a transaction is processed. Alternatively one could think of having a fee set by the app as long as the dedicated computing power is sufficient and an additional fee chosen by the users once the computing power is exhausted.
4. *Flat Fees:* The app might also charge a flat fee from the user. The flat fee would allow the user to invoke the app as much as he wants for a given amount of time (again, we might allow users to add tips to individual invocations of the app in case the miners cannot process all submitted transactions at a time).

The pricing mechanisms target different kinds of users and might be combined within a given app. For example, transaction fees are for users that only want to make one or two transactions now and then, while staking is appealing to long-term users.

1.3 Incetivizing App Deployment (and Staking)

Even though app developers might derive utility from using their app, it might seem appropriate that they also get rewarded when other users use it. Stakers also need to be incentivized; likely there will be competition among the apps for stake.

If the app uses fees to allocate computing resources among users, then the raised PHA or ATOs could go directly to the developer and the stakers, where the developer is free to determine the share that goes to him and the share that goes to stakers. Among the stakers, the proceeds are distributed proportionally to their stake. If the app has its own designated ATO, then the deployer and the stakers can either keep their earned ATOs staked, sell them on a secondary market, or redeem them anytime.

If user resource usage is allocated through (second-layer) staking of PHA or ATO, the staking module might "tax" part of stake; i.e., some of the staked PHA or ATOs go to some form of app treasury. From this treasury, the developer again specifies how much he gets to keep and how much is distributed proportionally among the (first-layer) stakers.

2 Further Considerations

2.1 Design Considerations

There are a couple of dimensions that need to be considered when designing a pricing mechanism for computing resources. They include:

1. *Incentives*: The pricing mechanism needs to provide sufficient incentives for the developers to deploy smart contracts. Also, staking must be incentivized, as there will be competition between apps for stake.
2. *Useability*: The pricing mechanisms should be tailorable to the needs of the users, and be as simple as possible.

3. *Recognizeability*: At best, the mechanism mimics pricing mechanisms that the market participants are used to from other networks.
4. *Robustness to price volatility*: It should not be the case that incentives of the users and the developers depend too much on the price level of PHA.
5. *Sustainability*: The design should generate sufficient demand for PHA so that the payouts made to the miners generate enough revenue to cover their costs.

I believe that the proposed mechanisms meet these criteria. Point 1 (*Incentives*) is covered in Subsection 1.3 above. *Useability* is achieved by leaving the app developers some choice in how they want to set up the pricing within their apps. In that way, they can tailor their pricing to the needs of their users.

Recognizeability is given, too. Staking funds to consume resources is the very idea of parachain auctions in the Polkadot. Transaction fee-based pricing models are very common, too.

The staking approach to allocating resources is also *robust to price volatility*, because the share of resources a user obtains is proportional to his staked funds, and users are affected symmetrically by price movements. So, a sudden spike in the price of PHA should not crowd out less wealthy users.

The last point, *sustainability* of the mechanism is an empirical question that can only be settled once Phala runs the mechanisms for a while. However, theoretical arguments are supporting a staking-based mechanism, which I discuss in the next subsection.

2.2 Theoretical Considerations

The PHA tokens serve a two-fold role in the Phala network. On the one side, they are paid out to miners for providing computing resources. On the other hand, they are required by users to consume these resources.

A sustainable tokenomics design needs to balance the incentives of the two sides. Miners require a sufficiently high PHA price so that they can cover their infrastructure costs by selling PHA. Users, on the other hand, might not want to buy enough PHA if the price is too high. In equilibrium, the resulting PHA price needs to be such that both sides obtain sufficient utility to participate.

I have analyzed a general version of this problem in my working paper "Utility Token Design".[3] I believe that the arguments in that paper can be adapted to the case of Phala. The general conclusion would be that existence of an equilibrium is not a problem. That is, theoretically, a design of staking on both sides will work. The tokenomics whitepaper for Phala would include a brief section outlining these arguments.

[3] Häfner, Samuel, Utility Token Design (November 20, 2021). Available at SSRN: https://ssrn.com/abstract=3954773.

3 Milestones and Roadmap

The RPF asks for the following milestones:

1. Complete the first version of the demand-end tokenomic design.
2. Publish the full tokenomic whitepaper.
3. Work with the dev team to implement the tokenomics, propose as a Khala runtime upgrade, and get it passed.
4. Run the tokenomic on Khala for 3 months and then improve based on the feedback, and launch it on Phala Network.

The completion of the first version of the demand-end tokenomic design will require intensive discussions with the Phala team to improve my understanding of Phala and to see what is technically feasible. This step also requires doing research on how other projects do it and on the adaption of my "Utility Token Design" paper. Once this is done, I expect the write-up of the full tokenomics paper to be fairly quick. As regards the third step (i.e., the implementation), I am lacking the required experience to estimate the required time. In short, I roughly expect the following roadmap.

	Milestone	Time
1.	Draft of Tokenomics Whitepaper	2 months
2.	Full Version of Tokenomics Whitepaper	1 months
3.	Implementaion	2 month
4.	Testing	3 months

Table 1: *Anticipated Roadmap*