IQ PROTOCOL: DEFI FRAMEWORK FOR SUBSCRIPTION BASED SERVICES AND RISK-FREE COLATERALLESS UTILITY LOANS

ANATOLY RESSIN, SIMON YAKUNIN, ALAN DURNEV

Abstract. In recent years, subscriptions have emerged to become one of the most popular ways in establishing economical relations between sellers and consumers. Instead of purchasing individual items on a regular basis people tend to pay in advance to have the right to consume a particular bandwidth from the service provider.

We are proposing a novel way to avoid some known pitfalls [1] in the token issuance process for service based companies where consumption is measurable over time and volume. Our targeted services are expected to work under a subscription model, and the volume of the services being delivered to the end-user depends on some initial (possibly renewable) agreement with the service provider. Examples of such services can be: File storage provider, IPFS Pinning Service, Internet Services Provider, Blockchain Monitoring Service, and many others: even traditionally off-chain services like a daily delivery of fresh fruits.

We propose a generic way for implementing subscriptions on-chain in a flexible and cheap way, while preserving all important workflows like cancel/refund policies, different time-frames, consumption rate limiting, overuse quota, discounts, etc. For that we introduce the concept of PowerTokens, that are utilized not as a means of payment, but rather as a deterministic over-time "energy" generator. Here, energy plays a role in accounting for the unit of service consumption (like gas units in Ethereum).

According to our proposal, the service provider serves its customers not on the basis of how many PowerTokens they have, but rather on the basis of how much energy they've accumulated while holding their PowerTokens. Consumed energy is tracked purely off-chain with a proposed new type of state channel: DvP — Delivery vs. Payment Channels. In this paper we show that DvP channels can be provably constructed and updated only on the basis of the publicly observable state of the blockchain and don't involve any transactions on the blockchain. Thus, they are more effective than traditional State Channels.

As well, we show how DvP channels allow for the consumption of centralised services in a purely decentralised and anonymous way. Here, users can receive services without registration and without sending any blockchain assets to the company, while both parties can deterministically infer user's eligibility (including dynamic limits) for receiving services.

The proposed tokenomics model makes it possible to organise a completely decentralised protocol for collateralless borrowing of PowerTokens that is at the same time is risk-free for the loan side of the protocol.

1. Introduction

1.1. **Tokenomics for off-chain services.** Creating sustainable tokenomics for an off-chain service is a known challenge. By the term *IQ Protocol off-chain service* we mean a service, that has well-defined utility, where this utility is not directly verifiable on chain, because it often deals with off-chain data, that is impractical to oraclize (e.g. due to the private nature of the delivery of the result). The fever of the 2017 ICOs has provided us many examples [2] that even if the service is well defined, and anticipated by users, it is extremely hard to create balanced tokenomics for it.

1.2. **Chicken-egg problem.** ICO/IEO/ILO are traditionally used to attract funds in advance. Then attracted funds are used to implement some vision that will make issued tokens meaningful (attach a utility). However, according to most regulations [3] [4] [5], sold tokens from day-zero should be equipped with some well-defined utility. Otherwise they could be treated as unregistered securities. So often token issuers are locked in a chicken-egg problem: to attach a utility for sold tokens one needs to implement some basic functionality, to cover development of such functionality one needs to sell tokens. Different associated problems were studied in [6]. In most cases the first and the only utility of an issued token is ability to pay for services, often with a discount if other means of payment are implemented as well. This approach is often referred to as *AppCoins* [5]. Unfortunately this utility is very unstable, because of the following reasons.

1.3. **Pitfalls of Tokens as AppCoins.** Upon issuance, AppCoins are often distributed between several groups of holders with different expectations and strategies:

- *Utility users* — users, who are interested in the service itself. These users could be potential service consumers as well as service providers in case where the service plays the marketplace role. Utility users are interested in *stability* of the token value, at least in terms of the service volume attached to one token, because it allows one to predict spendings/earnings and plan a budget. *Utility users tend to perceive tokens as private company money and a unit of accounting*;
- *Investors* — users, who are supporting service ideas, can help with co-marketing by spreading their inspiration among other potential investors and utility users. Investors are rationally interested in *growth* of the token value. *Investors tend to perceive tokens as company stock/shares*;
- *Traders/Speculators* — users, who are mostly indifferent to the service idea itself. Traders are primarily interested in token as a *tradeable asset*. Traders are mostly interested in token *volatility*. Traders can play a positive role in tokenomics by connecting different exchanges into a single token circulation ecosystem via arbitrage. However, aggressive traders sometimes participate in various price manipulations schemes like Pump&Dump,

that destabilize token value and can harm the two previously mentioned categories of holders.

The discrepancy between interests of token holders often prevents a token from reaching an equilibrium during the price finding process. This leads to high volatility of the token price. Thus, the token fails to be a unit of accounting. In these circumstances one of the worst steps that could be made by a company is to accept its coins for services using a floating rate between the AppCoin and some fiat currency. By doing this, the company removes the anchors on why the token should have any particular price. In the long term, it leads to a free fall of the token price [1].

We propose a possible solution which gives each type of user what he/she wants: protect utility users from volatility, while at the same time ensuring earnings for investors while providing some volatility to traders via the IQ Protocol.

1.4. **Target Services.** The IQ Protocol tries to establish sustainable tokenomics for services where consumption is measurable over time. These services are typically expected to work under a subscription model, and the volume of the services being delivered to the end-user depends on some initial (possibly renewable) agreement with the service provider. Examples of such services can be: File storage provider, IPFS Pinning Service, Internet Services Provider, Blockchain Monitoring Service, and many others: even traditionally off-chain services like a daily delivery of fresh fruits.

1.5. **Main Idea of IQ Protocol.** For every subscription based service, we propose two steps in building sustainable tokenomics:

- Assign a **Life-Time-Value** meaning to your tokens. From that moment onwards, you are not asking users to pay for your services with your tokens. Instead, you are asking users to hold your tokens as long as they want to utilize the service provided. In general, the more tokens a user holds, the bigger the volume of the service the user can consume. Volumes which correspond to a given amount of tokens being held are subject to change and that should be clearly stated in your terms and conditions. However, we recommend a natural ratio: total supply of your tokens should be at least partially covered by your capacity as a service provider. Partial coverage is acceptable because part of the tokens always will be locked in exchanges and used for trading rather than for receiving utility.

- Create a **Renting Pool** that works as follows. Anybody who has Life-Time-Value tokens can securitise them by providing tokens as liquidity to the Renting Pool and in turn, receive shares of the pool in form of *interest-bearing tokens*.

Afterwards, anyone who wants to consume your services will have two options: either buy your original tokens that have life-time value, or rent your tokens from the renting pool. The main idea here is that the original tokens are not released from the renting pool. Instead, the pool mints an expirable version of these tokens. The expiration date is requested by the borrower, and affects the upfront interest payment. An expirable version of these tokens are implemented using Non-Fungible Tokens (NFT). Until tokens are expired they are treated as a permission to consume a particular amount of service and are recognised by the service provider.

As tokens automatically expire, borrowers will not have the ability to cheat. Therefore, we are not asking for any collateral for this type of borrowing. All interest paid to the protocol is distributed across the lenders.

However, interest from a particular loan is not paid immediately to the lenders. It is streamed into the pool using a modified money streaming algorithm [7] exponential streaming. That type of money streaming allows for the resolution of two problems:

- If the borrower realised that he/she doesn't need a loan anymore, he/she is able to prematurely burn the loan and obtain a refund. At that moment, tokens are unlocked in the pool and become borrowable again.

- Exponential streaming corresponds to compound interest calculation strategy that prescribes to move a majority of the interest to earlier dates, rather than uniformly distribute it over the period. This prevents the borrower from cheating with floating interest rate. Because if we use linear interest streaming, changes to the interest rate will lead the borrower to burn the current loan and re-borrow with a better interest rate. Otherwise, he/she simply keeps the loan untouched. That introduces an asymmetry towards the borrower, giving him better conditions in comparison with lenders. Exponential streaming reduces the average game-theoretic preferences back to zero (honest game).

2. Assumptions

2.1. **Target Ecosystem.** IQ Protocol is being described in terms and standards that are traditional to the Ethereum ecosystem. Namely we refer to ERC20, ERC1155, and other Ethereum standards for tokens. These standards are almost automatically adopted by EVM-compatible blockchains like BSC, MoonBeam, Ethermint and others. Nevertheless, the definition of the protocol semantics is formulated in blockchain-agnostic way, that gives an ability to implement corresponding logic in various L1 protocols.

Moreover, all algorithms used in this paper *do not assume* Turing Completeness of the underlying execution engine. While in some numerical algorithms we use formally unbound yet quickly converging loops, one can easily unwind them manually into $O(1)$-time bounded sequences of operations. This fact allows the IQ Protocol to be implemented using more restricted environments like Solana BPF.

2.2. **Involved Parties.** We assume the presence of two groups of roles:

2.2.1. *Utility Roles:*

- *Service Providers* – selling their services under a subscription model, that has a predictable cash flow and will allow them to cover expenses as well as earn some margin.
- *Service Consumers* – buying services from service providers under predictable prices.

2.2.2. *Interest Roles:*

- *Enterprises* – attracting initial capital to establish a service providing business and earn money on top of it. Enterprises may own, maintain or orchestrate several different services.
- *Investors/Liquidity Providers* – bringing their capital to an enterprise in a trust-less way to be able to participate in its tokenomics and earn interest from it.

We will show how Service consumers could be interpreted as *Collateral-less Borrowers*, and how Liquidity Providers could be interpreted as *risk-free Lenders*.

3. Proof of Hold

3.1. **Generic model.** As mentioned before, IQ Protocol prescribes serving customers on the basis of holding corresponding tokens. In this section we will define the notion of *holding* in a non-formal way and then provide several quantitative formalizations.

To determine the amount of services that account a can consume at time t, we define *proof of hold* as a function $h(a,t)$ that can be computed over the history $\tau \in (-\infty, t)$ of account's balance and can simultaneously depict the amount of asset being held and the length of the period it is being held.

One of the functions that could serve as a proof of hold is a simple integral of the user balance over time (1).

$$h(a,t) = \int_{-\infty}^{t} \mathbf{balance}(a,\tau)d\tau, \tag{1}$$

where $\mathbf{balance}(a,t)$ – is the function that returns the balance of an account a at the specified time t.

The equation (1) represents the *unbound* proof of hold, i.e. it can grow indefinitely over time while the account holds a finite amount of assets. It corresponds to a rather unrealistic scenario where the service providers agree to serve ex-holders of an asset no matter how much time has passed since they actually held an asset.

> **Note:** We define a *subscription as a right to obtain a limited amount of services during a given time at an agreed frequency.*

According to this definition, if the service was not consumed timely, then the subscriber loses the right to reclaim it later. Thus, proof of hold for such subscriptions should be *bound*, i.e. not grow indefinitely.

To achieve this, we should apply a weighting function that will decrease the significance of "too old balances".

Applying weights to a function is essentially a convolution (2).

$$h(a,t) = \int_{-\infty}^{t} \mathbf{balance}(a,\tau)w(\tau - t)d\tau, \tag{2}$$

where $w(t)$ is the weighting function.

Let's consider two weighting functions (3,4) that correspond to different approaches to balance aging.

$$w_s(t) = \begin{cases} 1/s, & \text{if } -s \le t \le 0, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

$$w_\lambda(t) = \begin{cases} \lambda e^{\lambda t}, & \text{if t} < 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

In (3), weighting function $w_s(t)$ corresponds to the Simple Moving Average (SMA) with window size s. SMA treats equally all balances that are not older than s time units (e.g. seconds, blocks, ...) and ignores the other ones.

In (4), weighting function $w_\lambda(t)$ corresponds to the Exponential Moving Average (EMA) with parameter λ. EMA accounts for potentially an infinite amount of balances, assigning them exponentially decreasing weights in the direction of older ones.

3.2. **Proof of hold implementation.** Despite EMA (4) combining an infinite amount of historical balance values, it can be efficiently implemented as a time-driven value using $\mathcal{O}(1)$ memory cells without requiring to check actual history. In contrast, SMA (3) with time window s requires $\mathcal{O}(s)$ memory cells and implies checking historical data.

As we can see from the image (3.2), SMA and EMA basically share the same properties and both could be used to calculate proof of hold. However, EMA provides other useful properties that we will cover in the next sections, that makes it a preferable choice for the IQ Protocol.

Combining 2 and 4 we get (5).

$$h(a,t) = \lambda \int_{-\infty}^{t} \mathbf{balance}(a,\tau)e^{\lambda(\tau - t)}d\tau, \tag{5}$$

actually the term $(\tau - t)$ in (5) slides from $-\infty$ to 0, which means that $e^{\lambda(\tau-t)}$ slides from 0 to 1. In real setting $-\infty$ time just means genesis block, or even an *asset-genesis* (corresponding smart contract deployment block). As well it is assumed that before an asset-genesis block all balances are equal to zero.

In practice, $\mathbf{balance}(a,t)$ represents a piecewise constant function that changes only at specific points of time while remaining constant in between them. Let's consider interval $t \in [t_i, t_{i+1})$ on which $\mathbf{balance}(a,t)$ is constant and is equal to $\mathcal{B}_i(a)$. Then, we can write down a recursive computation scheme (6).

$$h(a,t_{i+1}) = \mathcal{B}_i(a) - (\mathcal{B}_i(a) - h(a,t_i))e^{-\lambda(t_{i+1}-t_i)} \tag{6}$$

Note that formula (6) remains valid if rewritten for any $t \in [t_i, t_{i+1})$, as (7).

$$h(a,t) = \mathcal{B}_i(a) - (\mathcal{B}_i(a) - h(a,t_i))e^{-\lambda(t-t_i)} \tag{7}$$

As it can be observed from equation (7), proof of hold $h(a,t)$ asymptotically approaches $\mathcal{B}_i(a)$. The nature of this convergence could be described either in terms of exponential approaching $e^{-\lambda t}$ or in more convenient terms of *gap-halving period* $T_{1/2}$. Those are two equivalent forms

of the same convergence law and can be defined by simple substitution (8).

$$e^{-\lambda t} = \left(\frac{1}{2}\right)^{\frac{t}{T_{1/2}}}, \quad \lambda = \frac{\ln 2}{T_{1/2}}. \tag{8}$$

While λ-exponential form is convenient for mathematical proofs, the gap-halving period formalization is more suitable for mass audience explanation. For the purposes of brevity, we will introduce an exponential extrapolation function (9) that allows finding the value that is asymptotically approaching X^* starting from X after time Δt with a gap-halving period $T_{1/2}$.

$$s(X, X^*, T_{1/2}, \Delta t) = X^* - (X^* - X)\left(\frac{1}{2}\right)^{\frac{\Delta t}{T_{1/2}}} \tag{9}$$

Equation (9) allows us to rewrite (6) as (10).

$$h(a, t_{i+1}) = s(h(a, t_i), \mathcal{B}_i(a), T_{1/2}, t_{i+1} - t_i) \tag{10}$$

3.3. **Account state snapshot.** Let's introduce the notion of *state snapshot* of account a at time t_i (11).

$$\boldsymbol{\sigma}_{t_i}[a] = \langle \mathcal{B}_i(a), h(a, t_i), t_i \rangle \tag{11}$$

Thus, $\boldsymbol{\sigma}[a]$, comprises the following list of fields:

- **balance:** A scalar value equal to the number of smallest indivisible units of an asset that was registered at time t_i. For account a in its state snapshot $\boldsymbol{\sigma}[a]$, this would be formally denoted $\boldsymbol{\sigma}[a]_{\mathrm{b}}$.
- **proofOfHold:** A scalar value equal to the proof of hold scoring defined by equation (6). Formally denoted $\boldsymbol{\sigma}[a]_{\mathrm{h}}$.
- **timestamp:** The timestamp at which balance $\boldsymbol{\sigma}[a]_{\mathrm{b}}$ and proof of hold $\boldsymbol{\sigma}[a]_{\mathrm{h}}$ were registered. The timestamp is formally denoted $\boldsymbol{\sigma}[a]_{\mathrm{t}}$.

Equation (6) opens up a possibility to implement account state snapshot updating within the same transaction that changes the balance. Let's consider a transaction T that transfers asset amount x from account a_1 to account a_2 at time t (12).

$$T = \langle a_1, a_2, x, t \rangle \tag{12}$$

Before applying a transaction, account a_1 has a state $\boldsymbol{\sigma}[a_1]$, where $t_{a_1} = \boldsymbol{\sigma}[a_1]_{\mathrm{t}}$ is the time at which account a_1 was updated. Same applies to account a_2. Account states will be updated according to the equations (13).

$$\begin{aligned} \boldsymbol{\sigma}[a_1] &:= \langle \boldsymbol{\sigma}[a_1]_{\mathrm{b}} - x, s(\boldsymbol{\sigma}[a_1]_{\mathrm{h}}, \boldsymbol{\sigma}[a_1]_{\mathrm{b}}, T_{1/2}, t - \boldsymbol{\sigma}[a_1]_{\mathrm{t}}), t \rangle, \\ \boldsymbol{\sigma}[a_2] &:= \langle \boldsymbol{\sigma}[a_2]_{\mathrm{b}} + x, s(\boldsymbol{\sigma}[a_2]_{\mathrm{h}}, \boldsymbol{\sigma}[a_2]_{\mathrm{b}}, T_{1/2}, t - \boldsymbol{\sigma}[a_2]_{\mathrm{t}}), t \rangle. \end{aligned} \tag{13}$$

3.4. **Transaction- vs time-driven values.** Blockchain-like systems are traditionally defined via transaction based state evolution [8]. New state emerges only as a result of applying a transaction to the previous state. Account balances are essentially part of the blockchain state. So we can consider a sequence of balance values $(b_0, b_1, b_2 \dots)$ (often b_0 being equal to 0) as values of $\mathbf{balance}(a, t)$ at time $t \in (t_0, t_1, t_2 \dots)$ where t_i is time at which the balance changing transaction was applied to the state. Without loss of generality, we state that there were no balance changing transactions between t_i and t_{i+1}. Let's call b_i the *transaction-driven value*.

We define the *time-driven value* as a pure function of blockchain state and physical time. We assume that physical time is universally synchronised[1] between all observers of the blockchain state. Thus, time-driven values will synchronously evolve across the globe as the physical time flows without applying any transactions to the state. Time-driven values are not persisted in blockchain, yet can be calculated on-demand using Smart Contract logic in read only mode.

The principle of time-driven values was previously used in blockchain projects, like Compound [9], Sablier [10] [7] and others. IQ Protocol also leverages time-driven values for the proof of hold calculation.

3.5. **Time-driven account state.** It is possible to extrapolate account state to any future point in time starting from the present moment even if there is no transaction that changes the account state. We can define a virtual transaction that moves 0 amount of asset from the same account to itself at time t assuming that the last account state update was at time t_a. Using equations (13) we get *time-driven account state* (14).

$$\widetilde{\boldsymbol{\sigma}}_t[a] = \langle \boldsymbol{\sigma}_{t_a}[a]_b, s(\boldsymbol{\sigma}_{t_a}[a]_h, \boldsymbol{\sigma}_{t_a}[a]_b, T_{1/2}, t - t_a), t \rangle \tag{14}$$

Note, that (14) is a pure function of blockchain state and the time, while it perfectly matches to initial continuous equation (5).

$$\widetilde{\boldsymbol{\sigma}}_t[a]_h = \lambda \int_{-\infty}^{t} \mathbf{balance}(a, \tau) e^{\lambda(\tau - t)} d\tau \tag{15}$$

where $\lambda = \frac{\ln 2}{T_{1/2}}$.

Using the fact that EMA is a linear operator [11], we can prove that total value of time-driven proof of hold across all accounts will be equal to EMA of asset's total supply over time (16).

$$\sum_{a \in A} \widetilde{\boldsymbol{\sigma}}_t[a]_{\mathrm{h}} = \lambda \int_{-\infty}^{t} \left(\sum_{a \in A} \boldsymbol{\sigma}_\tau[a]_{\mathrm{b}}\right) e^{\lambda(\tau - t)} d\tau \tag{16}$$

According to (16) we can conclude that if the total supply of an asset is fixed then total proof of hold across all accounts will converge to the same amount as total supply.

4. Power Tokens

4.1. **Proof of Hold as accumulated Energy.** One important analogy that could be used to interpret proof of hold $\widetilde{\boldsymbol{\sigma}}_t[a]_{\mathrm{h}}$ is *normalized accumulated energy* $\widetilde{E}(t)$. The balance $\widetilde{\boldsymbol{\sigma}}_t[a]_{\mathrm{b}}$ plays the role of *power* $P(t)$ and $e^{-\lambda} = \left(\frac{1}{2}\right)^{\frac{1}{T_{1/2}}} = q \in (0, 1)$ is energy retention ratio per time unit. For example $q = 0.3$ means that only 30% from accumulated energy remains available for the next step and the other 70% dissipates. Like in physics, we define power as an ability to generate energy per time unit. In a discrete setup, where time is advanced by atomic time units (blocks, slots ...) which we denote by $\mathbf{1}$, the evolution of energy $E(t)$ could be described by the difference equation (17).

$$E(t + \mathbf{1}) = P(t) \cdot \mathbf{1} + qE(t) \tag{17}$$

[1] Ignoring relativistic effects

In further equations we omit **1**, as numerically it is equal to one.

It is easy to see that (17) can be rewritten for arbitrary time shift as (18).

$$E(t+n) = q^n E(t) + \sum_{k=0}^{n-1} q^k P(t+k) \tag{18}$$

If $P(t) = P(t_0)$ for every $t \in (t_0, \infty)$ then we can reduce (18) to (19).

$$\begin{aligned} E(t_0+n) &= q^n E(t_0) + P(t_0) \sum_{k=0}^{n-1} q^k \\ &= q^n E(t_0) + P(t_0) \frac{1-q^n}{1-q}. \end{aligned} \tag{19}$$

Let's denote the limit $\lim_{n\to\infty} E(t_0+n)$ as $E^*(t_0)$.

$$E^*(t_0) = \lim_{n\to\infty} E(t_0+n) = P(t_0)\frac{1}{1-q} \tag{20}$$

Equation (19) could be rewritten using (20) as (21).

$$E(t_0+n) = E^*(t_0) - (E^*(t_0) - E(t_0))q^n \tag{21}$$

We can see as well from (20) that asymptotic value $E^*(t_0)$ is strictly proportional to $P(t_0)$. If we denote a normalized version of energy $\widetilde{E}(t) = (1-q)E(t)$ then we see that $\widetilde{E}^*(t_0) = P(t_0)$ and the equation (21) being normalized can be written as (22).

$$\begin{aligned} \widetilde{E}(t_0+n) &= \widetilde{E}^*(t_0) - (\widetilde{E}^*(t_0) - \widetilde{E}(t_0))q^n \\ &= P(t_0) - (P(t_0) - \widetilde{E}(t_0))q^n \\ &= s(\widetilde{E}(t_0), P(t_0), T_{1/2}, n), \end{aligned} \tag{22}$$

where $T_{1/2} = (\log_{\frac{1}{2}} q)^{-1}$ that fully corresponds to (10). Thus we can state that our definition of proof of hold can be interpreted in terms of normalized accumulated energy where balance of an account corresponds to power that produces energy over time.

In the following section we will show how accumulated energy could be tokenized effectively creating a brand new class of assets with unique properties.

4.2. **Transferability of Proof of Hold.** In previous sections we defined proof of hold as a function of historical balances (namely EMA). Here we extend this definition by allowing the transfer of proof of hold, explain why it could be needed, and explore its prerequisites and implications.

The original definition of proof of hold allows everybody to easily (in $\mathcal{O}(1)$) inspect the recent history of balances in an aggregated way. IQ Protocol tokenomics model prescribes to use this value as eligibility metrics for receiving services.

Service providers can ask a consumer for a particular value of proof of hold to deliver its services. However, this concept does not specify any rate-limiting logic of service requests. We propose two mutually exclusive approaches on how to address this problem and one that virtually combines the benefits from both of them:

- Throttling consumer requests according to his/her proof of hold;
- Using proof of hold as currency and pay with it.
- DvP channels

4.2.1. *Proof of Hold based throttling.* In this scenario, we assume that proof of hold is calculated only on the basis of historical values of balances and *never can be changed in any other circumstances, especially sent to an arbitrary address.* In that case service provider could implement a simple rate-limiting policy[2]:

- Receive a request at time t signed by account a;
- Retrieve account's a proof of hold $\boldsymbol{\sigma}_t[a]_{\mathrm{h}}$;
- Calculate throttling threshold $\Delta\tau$ that is inversely proportional to $\boldsymbol{\sigma}_t[a]_{\mathrm{h}}$
- Check whether the last request time $\mathbf{last}[a]$ from the account a is older than $t_{now} - \Delta\tau$
  - If $\mathbf{last}[a] < t_{now} - \Delta\tau$ or $\mathbf{last}[a]$ is not defined - then service the request and update $\mathbf{last}[a] := t$;
  - Otherwise refuse the request.

As it can be seen from above, this policy is based on proof of hold checks rather than account balance checks. This mitigates a *ping-pong attack* that is applicable to systems with transferable access rights. In this attack, malicious consumers are trying to trick the service provider by transferring access rights back and forth between themselves to obtain twice more in volume of service than they are eligible for. This attack is ineffective if the service provider implements the proof of hold checking policy before servicing a client.

> **Note:** *It shows that our definition of proof of hold allows the existence of transferable fungible assets that represent access rights and are ping-pong-attack-proof.*

4.2.2. *Proof of Hold as a spendable asset.* Unlike the previous approach where proof of hold was used by service providers together with external data (like $\mathbf{last}[a]$) to limit the consumption of service, here we propose a technique how proof of hold can be used alone to bring sufficient information about usage limits. For that we will detach the asset's proof of hold from the asset itself and will represent proof of hold as a separate (yet dependent) asset.

This new asset has a time-driven balance that evolves exactly like (9) with addition that *it can be changed directly by an account owner in a total amount preserving manner.*

We allow the transfer of computed proof of hold from the original account to another account without transferring the corresponding balance. At the same time we guarantee that the total proof of hold on both involved accounts remains the same as if no transfer would take place. Thus keeping the total proof of hold in a system according to the equation (16).

> **Note:** *In this setup, the only allowed way to consume a service is to send some amount of proof of hold to the predefined service provider's account. Amount of proof of hold being sent corresponds to the amount of service volume being requested from the provider. Without that constraint, detaching proof of hold from*

[2]This logic can be implemented on-chain for decentralized service providers or off-chain for centralized ones.

the primary asset makes the system vulnerable to the ping-pong attack.

As of now, we will use the term *energy* (as in 4.1) to denote transferable proof of hold. By analogy with transaction defined in (12), we also introduce *energy transaction* (23) that moves part of proof of hold (interpreted as energy).

$$T_E = \langle a_1, a_2, \Delta E, t\rangle, \tag{23}$$

where a_1 and a_2 are accounts with states $\boldsymbol{\sigma}_{t_{a_1}}[a_1]$ and $\boldsymbol{\sigma}_{t_{a_2}}[a_2]$ respectively, ΔE is transferred energy and t is transaction time. In this case, the state update rule will look like (24).

(24)
$$\boldsymbol{\sigma}[a_1] := \langle \boldsymbol{\sigma}[a_1]_\mathrm{b}, s(\boldsymbol{\sigma}[a_1]_\mathrm{h}, \boldsymbol{\sigma}[a_1]_\mathrm{b}, T_{1/2}, t - \boldsymbol{\sigma}[a_1]_\mathrm{t}) - \Delta E, t\rangle,$$
$$\boldsymbol{\sigma}[a_2] := \langle \boldsymbol{\sigma}[a_2]_\mathrm{b}, s(\boldsymbol{\sigma}[a_2]_\mathrm{h}, \boldsymbol{\sigma}[a_2]_\mathrm{b}, T_{1/2}, t - \boldsymbol{\sigma}[a_2]_\mathrm{t}) + \Delta E, t\rangle.$$

Similarly to energy transaction we define *power transaction* (25) as an exact reinterpretation of balance changing transaction (12),

$$T_P = \langle a_1, a_2, \Delta P, t\rangle, \tag{25}$$

where ΔP is transferred power. State update rules are the same as in (13).

Transfer can be explained using the mental model of a power generator that produces accumulatable yet dissipating energy over time. In this model, the generator's power corresponds to the account's balance, the accumulated energy corresponds to the account's proof of hold, and the transfer of energy means the transfer of proof of hold. An important addition to this interpretation is treating energy as a *fungible* and *transferable* asset that has a time-driven balance and depends on the primary (power) asset. Energy itself is a dissipating asset, the balance of which dissolves gradually over time in case of absence of the power asset. Every time unit the power asset generates an amount of energy that is proportional to the power balance. Thus, the energy balance is growing. However, because of a fixed energy dissipation ratio, the more energy is accumulated on the account - the more energy is dissipated (proportionally). This fact implies that when the absolute value of dissipated energy becomes equal to the absolute value of energy supplied by the power generator - the balance of energy stabilizes. Actually, in real-number arithmetic it will take an infinitely long time to converge, yet in finite-precision arithmetic it converges in reasonable time $\mathcal{O}(-\log \varepsilon)$.

What is happening, when we allow the transfer of energy, could be demonstrated on two accounts a_1 and a_2 where a_1 has non-zero power balance and almost full energy balance, whereas a_2 has both balances equal to 0. After transferring ΔE energy from a_1 to a_2, the energy balance of a_1 starts growing back its saturation level, while energy balance on a_2 starts decreasing due to lack of backing power. It is worth noting that the speed of a_1 energy balance replenishment is equal to the a_2 energy balance draining. In some situations it could be useful to think about this process as energy flowing back.

Summary: *We just introduced a pair of coupled assets: power-asset (primary) and energy-asset (secondary). Both power and energy are transferable. Power has a static nature - it remains the same unless power transactions are applied to the system state, whereas energy has a dynamic, time-driven nature: it is being generated in presence of backing power and is dissolving in absence of it. IQ Protocol prescribes to pay with energy while keeping power intact.*

4.2.3. *Consumption rate limits.* As follows from equation (22), energy growing speed is non-linear over time: it decreases proportionally to the elapsed time (in absence of disturbing factors like power/energy transactions). If energy spending transaction is applied to the account state, then assuming unchanged power, we get an increased energy growing speed.

Thus, if energy transfers are used for limiting access to the resources, then one can observe a discrepancy of total spent energy between cases with different service request frequency. The more frequent the requests – the more total energy which could be spent over the same time.

Nevertheless, the maximum consumption rate has an upper limit that can be estimated as follows. For simplicity, we will use a continuous time model. In that model, assuming constant power balance $\mathcal{B}$ that appeared at time $t = 0$, the energy evolution will be described by a formula (26), where λ is energy growth intensity.

$$E(t) = \mathcal{B}(1 - e^{-\lambda t}) \tag{26}$$

Let's consider a strategy where the user repeatedly waits time Δt and drains all the available energy. In this case, the consumption ratio $C(\Delta t)$ could be described by formula (27).

$$C(\Delta t) = \frac{E(\Delta t)}{\Delta t} \tag{27}$$

As it was mentioned, $C(\Delta t)$ grows when $\Delta t \to 0$. Thus, maximum consumption rate could be described by a limit (28).

$$C_{max} = \lim_{\Delta t \to 0} C(\Delta t) = \lim_{\Delta t \to 0} \frac{E(\Delta t)}{\Delta t} \tag{28}$$

Using the fact that $E(0) = 0$, we can rewrite (28) to (29).

$$C_{max} = \lim_{\Delta t \to 0} \frac{E(0 + \Delta t) - E(0)}{\Delta t} = E'(0) \tag{29}$$

Thus,

$$\begin{aligned} E'(t) &= \lambda \mathcal{B} e^{-\lambda t}, \\ C_{max} &= E'(0) = \lambda \mathcal{B}. \end{aligned} \tag{30}$$

Equation (30) shows that maximum possible energy consumption is proportional to both intensity λ and power balance $\mathcal{B}$. Let's recall that λ is inversely proportional to $T_{1/2}$ (8).

$$C_{max} = \frac{\mathcal{B} \ln 2}{T_{1/2}} \tag{31}$$

Note: We demonstrated a connection between gap-halving period $T_{1/2}$ and maximum possible consumption rate C_{max}. Formula (31) allows to reasonably choose $T_{1/2}$ on a basis of projected

maximum resource consumption rate associated with one power token.

At the same time, we see that using proof of hold as spendable energy lacks the property of countable additivity [12] over time, meaning that dividing your consumption into two separate portions is preferable over consuming energy at once. If service received after spending energy is virtual and does not imply any physical resource consumption (e.g. shotgun bullets fired in a video game) - this model could be preferable. From the other side, if consumption is tied to physical resources (e.g. CPU processing time, storage or even Ethereum-like Gas concept) users can expect additivity at least within a specified time-frame that require additional techniques.

5. Renting Pools

5.1. **Main concepts.**

5.1.1. *Asset transferability and transformation.* In smart contract enabled blockchains, it is common to wrap one asset into another (e.g. WETH as Wrapped Ethereum). Technically, the original asset is being locked (immobilized) on a smart contract and the other asset is being issued as a temporary replacement. Original assets are unlocked only when the replacement asset is returned and burnt. Provided that every unit of the replacement asset is backed by an immobilized original asset, we can talk just about the asset's transformation without losing mobility. Any rights attached to the original token could be preserved if there exists a formal way of tracking the replacement asset down to the original one. According to this view, IQ Protocol provides both transformation and tracking functionality for arbitrary[3] assets, while ensuring one important exception from usual crypto transferability rules. IQ Protocol defines a legitimate condition where your crypto can be confiscated - if it was borrowed for a fixed period and this period is now over.

5.1.2. *Connection with Power Tokens.* Power tokens described in (4), by definition can bring utility value without being spent. One can repeatedly extract value as long as power tokens reside on his/her account. Power token balance only determines the upper limit for the utility value extraction over time. Thus, the owner of the power token has two rights:

- To move power tokens;
- To extract value.

The right to move can be enforced by blockchain rules (e.g. smart contracts) in a decentralized, verifiable way. On the other hand, the enforcement of the right to extract value has more legal nature and is purely determined by the abilities of the underlying technology or obligations of the service provider (that is supposed to be the power token issuer). Sometimes, the result of the value extraction can be verified on-chain, possibly using zero-knowledge proof techniques. However, when this is not possible, it could be implemented using public offer and attached with Terms and Conditions. In any case, the primary source of economical value for a power token is derived from the demand of provided services.

The fact that a single power token can be used for extracting an infinite amount of value in the future makes it a valuable asset in the present. Its present value can be estimated using the traditional annuity formula.

$$PV = P \times \frac{1-(1+r)^{-n}}{r}, \tag{32}$$

where P is the maximum extractable value per period, r is the nominal period interest rate[4], n is the number of periods and in our case $n = \infty$, so that formula (32) is simplified as a result (33).

$$PV = \frac{P}{r} \tag{33}$$

For small r the present value of the power token could be sufficiently higher than the annual extracted value. So that for large volumes it could be reasonable to rent it, rather than buy. Separation of rights to move and extract value from power tokens is one step towards renting mechanics. Another step is to allow *wrapping* of the power token into a transparent expirable envelope. Expiration means that after a specified period, power tokens are automatically returned to the original owner. Transparency means that wrapped tokens are treated by the service provider in the very same manner, as unwrapped ones.

While generic expirable envelopes can be used for implementing peer-to-peer renting, we would rather propose the very same anonymous aggregated approach that was proposed in Compound [9] - *asset pooling*.

We decided to inherit Compound terminology of *lenders* and *borrowers* - to emphasize some important analogies with the credit market. At the same time, IQ Protocol cannot be qualified as a financial credit service.

By moving original tokens into a renting pool - lenders are providing their tokens in an aggregated way to borrowers for a limited time, so that borrowers would be able to extract utility value from the borrowed tokens during this time. Acquiring tokens from the pool could be treated as a *loan* that is taken on particular terms: duration and interest. In a general case, borrowed tokens are as mobile as original tokens. Although, they are packed into NFTs for simplicity of tracking and clean-up maintenance. In IQ Protocol there is no combination of principal and interest payments: interest is being paid upfront while the principal payment is paid after expiration. IQ Protocol uses *social incentivization* for returning the principal payment - a small part of the interest is being reserved for any party that will initiate a transaction after loan expiration. For the front-running prevention, IQ Protocol uses two grace periods: a short period when only the borrower can return the expired loan (T^{own}_{grace}) and an even shorter period when the renting pool owner can do that ($T^{pool-owner}_{grace}$). After both grace periods - anybody can perform the maintenance and grab the reward. All the interest except maintenance fee is being gradually transferred (*streamed*) to the renting pool reserves, thus increasing the total value of funds locked in the pool. This streaming represents the lender's profit.

Despite the similarity between IQ Protocol loans and loans in financial services - there are two main differences between them:

[3]Currently IQ Protocol supports only fungible assets, like ERC20.
[4]For reference value of r, one can use the annual deposit rates of USA and EU banks.

- IQ Protocol loans are guaranteed to be returned (risk-free), thus not requiring collateral;
- Upfront interest payment has fundamentally different origins, as it should reflect the market price for the services the loan has been borrowed for.

5.1.3. *Automated Interest Rate Discovery.* For sustainable renting pool functioning, it is crucial to find an optimal interest rate discovery strategy that will maximize lender's profit while ensuring sufficient liquidity level at the same time.

For services with *elastic demand*[5] the optimal interest rate can be obtained through a feedback loop that changes the *price* over time trying to maximize $profit = price \cdot demand$ using the gradient descent method [13]. Traditionally, it could be described by the differential equation (34).

$$\begin{cases} demand = f_{market}(price), \\ \frac{d}{dt}price = f_{control}\left(\frac{d}{d\ price}(demand \cdot price)\right), \end{cases} \tag{34}$$

where f_{market} is a market reaction function that describes behavior of demand in response to price change and $f_{control}$ is monotonically increasing function with $f_{control}(0) = 0$ that determines the sharpness of the price reaction to the profit change.

It is impractical to solve the equation (34) analytically, as the f_{market} function is not known and even could change over time and its form can be determined only as an input from oracles. Though, it can be relatively easily implemented as a recursive adaptive scheme. Finding appropriate $f_{control}$ is a subject of adaptive scheme as well. If $f_{control}$ slope is too high, then the process of finding the optimal price can diverge (starts oscillating). If the slope is too low, then the optimal price convergence process can take considerable time, thus resulting in loss of profits.

For the sake of simplicity, in the current version of the IQ Protocol we decided to put the adaptive scheme for the market related part off-chain. That is why we introduce $price_{base}$ (45) that is supposed to be set by oracles specific to particular services. In the simplest scenario, the $price_{base}$ is set by the owner of the renting pool to reflect fair market service price.

From one perspective, the best possible scenario for all tokens put into the renting pool - is to be borrowed. However, static service price even being fair, can lead to the renting pool being drained that is undesirable for two reasons:

- New service customers cannot technically borrow even if they agree to pay more;
- Liquidity providers cannot withdraw their funds (put into the renting pool) and accumulated rewards, so the asset that represents renting pool shares becomes illiquid.

It means that price discovery strategy should incorporate some *bonding curve* [14] related mechanics. Consistent loan price discovery strategy should follow two rules:

- *Bonding Rule.* Decreasing amount of tokens available for borrowing should increase the price of loans to disincentivize borrowers from further borrowing and incentivize lenders to refill the renting pool (provide more liquidity);
- *Additivity Rule.* Provided that all loans are borrowed sequentially and each one of them changes borrowing conditions for subsequent ones - there should be no price incentive either to split bigger loans to smaller chunks or to merge smaller ones into a bigger one.

Implementing the Bonding Rule for renting pools is even trickier than for CFMM/AMM [15]. Due to interpretation of the final price of the loan as a service subscription fee - its rapid increase in response to decreased liquidity will discourage potential service customers to subscribe or renew existing service subscriptions even before liquidity is approaching to the zero level. Since the subscription fee is the only source of profit for lenders - the rapid growth of the bonding curve leads to a poor *capital efficiency.* This way, the bonding curve should be relatively insensitive to available reserves drops down to levels that are considered dangerous for lenders where their shares become illiquid.

To formalize price discovery, we will operate with three amounts of tokens:

- R_{own} is current total reserves of the renting pool;
- R_{used} is currently borrowed amount of tokens;
- x is the amount of tokens to be borrowed.

Then we define *liquidity ratio* r as (35).

$$r = \frac{R_{own} - R_{used}}{R_{own}} \tag{35}$$

For the liquidity protection, we define a *critical liquidity level* r_{crit} as the level down to which liquidity ratio can fall during borrowing operations. It is important to note that r can fall below r_{crit} if lenders will withdraw their liquidity. Nevertheless, the price of loans is defined as $+\infty$ if $r \leq r_{crit}$. In such situation no borrowing operations are technically possible, unless sufficient liquidity will be either returned from expired loans or provided by new lenders.

On the interval $(r_{crit}, 1]$ the price of loans is continuous and has a vertical asymptote at point r_{crit}. The first step to define a price discovery strategy is *raw bonding curve* defined as function f_{raw} over r that should be equal to 1 at point $r = 1$, $+\infty$ on interval $[0, r_{crit}]$ and should have a controllable slope k on interval $(r_{crit}, 1]$. For the raw bonding curve we propose an equation (36). Let's consider r_{crit} and k fixed (defined on the level of the renting pool), so we will not include them as explicit parameters for defined functions.

$$f_{raw}(r) = \left(\frac{(1 - r_{crit})}{r - r_{crit}} - 1\right) \cdot k + 1 \tag{36}$$

The term "raw" means that this bonding curve is defined only for the situation when the actual value of r is obtained by borrowing a single loan from the pool with previous state $r = 1$ (100% liquidity available for borrowing). We need this clarification for defining a full form of bonding curve for which the additivity rule is held. Additionally, we define two helper functions h_{raw} (37) and h_{bond} (38). The h_{raw} function takes into account the borrowed amount x, yet it has meaning only if x is borrowed from the renting pool with $r = 1$. The structure of h_{bond}

[5]We say that service demand is elastic if the consumption volume is changed significantly due to the price change.

allows one to implement additivity within non-critical liquidity interval $r \in (r_{crit}, 1]$.

$$h_{raw}(R_{own}, x) = x \cdot f_{raw}\left(\frac{R_{own} - x}{R_{own}}\right) \tag{37}$$

$$h_{bond}(R_{own}, R_{used}, x) = \frac{h_{raw}(R_{own}, R_{used} + x) \; - \; h_{raw}(R_{own}, R_{used})}{x} \tag{38}$$

Now we can define a full form of bonding curve that can be used even in situations when the pool is partially utilized ($R_{used} \neq 0$) and has a guard against a critical level of liquidity (39).

$$f_{bond}(R_{own}, R_{used}, x) = \begin{cases} +\infty, & \text{if } \frac{R_{own} - (R_{used} + x)}{R_{own}} \leq r_{crit}, \\ h_{bond}(R_{own}, R_{used}, x), & \text{otherwise.} \end{cases} \tag{39}$$

Provided that fair market price for services is reflected by $price_{base}$ we can define a price for borrowing power tokens corresponding to the service. If user wants to borrow x power tokens for time Δt from the renting pool with own reserves R_{own} and already used reserves R_{used} - then the price of the loan could be described by $price_{loan}$ function (40).

$$price_{loan}(R_{own}, R_{used}, x, \Delta t) = x \cdot \Delta t \cdot f_{bond}(R_{own}, R_{used}, x) \cdot price_{base} \tag{40}$$

One can demonstrate that the function (38) ensures the Additivity Rule for the $price_{loan}$ (41).

$$price_{loan}(R_{own}, R_{used}, x + y, \Delta t) = price_{loan}(R_{own}, R_{used}, x, \Delta t) + price_{loan}(R_{own}, R_{used} + x, y, \Delta t) \tag{41}$$

Returning to the loan terminology, the price of the loan can be reformulated as *loan interest rate* normalized to one year APR (42).

$$APR(R_{own}, R_{used}, x, \Delta t) = \frac{price_{loan}(R_{own}, R_{used}, x, \Delta t)}{x \cdot \Delta t} \cdot \frac{1 \text{ year}}{\mathbf{1}} = f_{bond}(R_{own}, R_{used}, x) \cdot price_{base} \cdot \frac{1 \text{ year}}{\mathbf{1}}, \tag{42}$$

where $\mathbf{1}$ is one time unit used for measuring loan duration. Thus, $\frac{1\text{year}}{\mathbf{1}}$ represents time units per year. It may seem counter-intuitive for $price_{base}$ to be present as a factor in an expression, where all factors are expected to be dimensionless. Actually, $price_{base}$ is dimensionless as it is a quotient where numerator is an amount of power tokens representing service consumption rate and denominator is an amount of tokens in which service pricing is defined. Since both pricing tokens and power tokens that can be converted to each other using dimensionless rate - $price_{base}$ appears to be dimensionless as well.

Equations (40) and (42) are used for Automatic Interest Rate Discovery in IQ Protocol.

5.1.4. *Pool factory.* One of the benefits of IQ Protocol is the ability to reinterpret already existing tokens of some company as power tokens that unlock the access to services of that company. As a side effect, this demonstrates that power tokens are utility tokens. From now on, let's use the term *original token* to denote an already issued and existing token.

IQ Protocol itself is not created for a particular company and represents a decentralized platform that can be used by any company to create their own renting pools via an IQ Protocol pool factory smart contract. Creating a renting pool instance has both technical and legal implications for the company, as it prescribes to accept all power tokens derived from the original token and reflect that in the Terms and Conditions. It is important to note that IQ Protocol guarantees the total amount of power tokens in circulation to be always compensated with exactly the same amount of original tokens locked inside IQ Protocol. Thus, it makes it possible to treat power tokens as an altered version of original tokens without violating original token total supply promises for the community.

5.1.5. *Service provider incentive.* One can notice that interest payments earned by lenders are essentially an ordinary subscription fee that is usually paid to the service provider in traditional economics. It might seem irrational that actual work is done by the service provider, yet the rewards are received by other parties. However, the service provider could be one of the lenders and have a sufficient portion of shares in a renting pool. Thus, being able to cover its costs. This situation allows us to interpret lenders as shareholders of the virtual *enterprise*.

Another solution implies introducing direct deduction of service provider costs and margin from aggregated interest payments prior to any distribution. In this case, the service provider can abstain from having any nominal shares in the renting pool.

5.1.6. *Multi-service Pool as Enterprise.* Sometimes, the service provider uses shared facilities to provide different services. Such services can have different consumption models and different token holding requirements. At the same time, they could be dependent to the extent where it is reasonable to think about them in terms of *single aggregated capacity* of the service provider. IQ Protocol can reflect this situation by allowing the combination of different power tokens based on the same original token into a single renting pool. This is beneficial both for lenders and borrowers: lenders are providing liquidity without specifying an exact purpose how borrowers would like to use them, borrowers at the same time have the access to the flexible pool of liquidity (aggregated capacity).

5.2. **Renting Pool state.**

5.2.1. *High-level structure.* Let's define a high-level structure of renting pool state $\mathcal{P}$ (43).

$$\mathcal{P} = \langle \mathcal{I}, \mathcal{Q}, token_{liq}, \mathcal{T}_{accepted}, \mathcal{R}, conv, bonding \rangle, \tag{43}$$

where $\mathcal{I}$ covers all lender-specific functionality and data, $\mathcal{Q}$ (45) covers all service- and borrower-specific functionality and data, $token_{liq}$ is a reference to the original token that provides liquidity, described by renting pool reserve state $\mathcal{R}$ (48). $\mathcal{T}_{accepted}$ represents all tokens accepted as interest payment means for borrowing. Essentially $token_{liq} \in \mathcal{T}_{accepted}$. Accepted tokens are not limited to $token_{liq}$, however for the sake of auto-compounding all of the interest earned by the renting pool in any asset is automatically converted to liquidity token using the *conv* function (46). The *conv* function has both oracle nature and Decentralized Exchange (DEX) functionality, providing actual convert rates and executing

the conversion according to reported rates. Formally, we will use $y = conv(token_{from}, token_{to}, x)$ notation as a function when we have to perform only the estimation of a conversion. For the conversion itself, we will use **swap** $\langle token_{from}, x\rangle$ **to** $\langle token_{to}, y\rangle$ to perform swapping x $token_{from}$ tokens for y $token_to$ tokens.

The *bonding* part of $\mathcal{P}$ (51) describes the used bonding curve and its parameters.

5.2.2. *Lender's side* $\mathcal{I}$. Internally for tracking all the provided liquidity IQ Protocol uses the notion of *share*. Shares are abstract units that are used just for specifying granularity of ownership of the assets locked in the renting pool. Number of shares ω has meaning only in quotient with the total number of shares Ω.

$$\begin{aligned} \mathcal{I} &= \langle \Omega, stake, n_{stake}\rangle, \\ stake &: id \to \{\mathcal{S}_i\}, \\ \mathcal{S}_i &= \langle addr, \omega, amount_{stake}, t_{stake}\rangle \end{aligned} \tag{44}$$

When lender (liquidity provider) identified by address $addr$ provides $amount_{stake}$ of original tokens at time t_{stake} he/she receives the corresponding amount of shares ω packed into an NFT (id) with associated $\mathcal{S}_i$ data, where i is associated with id via $stake$ mapping. Identifiers id are issued by IQ Protocol sequentially, possibly hashing their sequential numbers and maintaining total number n_{stake} of issued NFTs.

5.2.3. *Borrower's side and Services* $\mathcal{Q}$. In the multi-service pool model we define a set of services $\mathcal{Q}$, where each service Q_i has its own unique power token PT that is required to be held in order to be served with service Q_i. In general case, IQ Protocol allows to peg service pricing to arbitrary token. We denote this token, as $token_{base}$. Renting one power token during one unit of time should be equal to $price_{base}$ nominated in $token_{base}$ if *pool liquidity ratio* is 0 (see bonding mechanics at 5.1.3). Service Q_i allows two different types of holding power tokens:

- *Own power tokens* obtained via wrapping liquidity token directly through service interface;
- *Borrowed power tokens* (loans) obtained via borrowing mechanism.

Loans can be either *refundable* or *non-refundable*. By the word refund, we denote returning a part of upfront paid interest in case the loan is returned earlier than its maturity time. Non-refundable loans can have any duration within a dur_{loan}^{nonref} range, defined per service Q_i. On the other hand, refundable loans must have duration within the discrete set of allowed durations T_{allow}^{ref} predefined on the level of renting pool (enterprise) (48).

Service performs tracking of issued loans via n_{loans} that increases every time the new loan is borrowed and is used for the calculation of an identifier of a corresponding NFT.

To cover service Q_i running costs, part of the interest payment is transferred to the service provider in proportion controlled by fee_{serv} parameter.

Power token PT is described by its gap-halving period $T_{1/2}$ and power/energy state mapping $\boldsymbol{\sigma}$ across all accounts.

$$\begin{aligned} \mathcal{Q} &= \{Q_i\}, \\ Q_i &= \langle PT, token_{base}, price_{base}, loan, n_{loans}, \\ &\quad dur_{loan}^{nonref}, fee_{serv}\rangle, \\ PT &= \langle T_{1/2}, \boldsymbol{\sigma}\rangle, \\ dur_{loan}^{nonref} &= [dur_{min}, dur_{max}] \end{aligned} \tag{45}$$

In addition to the already described parts of $\boldsymbol{\sigma}$ ($\boldsymbol{\sigma}[a]_{\mathrm{b}}$,$\boldsymbol{\sigma}[a]_{\mathrm{h}}$, $\boldsymbol{\sigma}[a]_{\mathrm{t}}$), we introduce an *own balance* $\boldsymbol{\sigma}[a]_{\mathrm{w}} \leq \boldsymbol{\sigma}[a]_{\mathrm{b}}$. The difference between $\boldsymbol{\sigma}[a]_{\mathrm{w}}$ and $\boldsymbol{\sigma}[a]_{\mathrm{b}}$ emerges because of loans L_i (47).

$$\begin{aligned} loan &: id \to \{L_i\}, \\ L_i &= \langle addr, amount_{loan}, t_{start}, t_{end}, token_{pay}, interest, fee_{gc}\rangle, \\ conv &: token \times token \times \mathbb{R} \to \mathbb{R} \end{aligned} \tag{46}$$

For convenience, we will use the names of the tuple parts as functions for extracting corresponding values from the tuple (e.g. $addr(L_i)$ will extract the $addr$ part of the L_i tuple). As well, let's introduce a function $range(L_i)$ that extracts t_{start} and t_{end} range from the L_i tuple.

$$\begin{aligned} range(L_i) &= [t_{start}, t_{end}], \\ \boldsymbol{\sigma}_t[a]_{\mathrm{b}} &= \boldsymbol{\sigma}_t[a]_{\mathrm{w}} + \sum_{i:(addr(L_i)=a)\wedge(t\in range(L_i))} amount_{loan}(L_i) \end{aligned} \tag{47}$$

Each loan is represented as an NFT (id) with associated data L_i, where

- $addr$ is an address of current NFT owner;
- $amount_{loan}$ is borrowed amount of power tokens;
- t_{start} is a timestamp of borrowing;
- t_{end} is a timestamp of loan expiration (maturity date);
- $token_{pay}$ is a token in which the loan interest is paid ($token_{pay} \in \mathcal{T}_{accepted}$);
- $interest$ is the loan interest;
- fee_{gc} is an incentivization fee for loan cleanup/maintenance.

Both $interest$ and fee_{gc} are paid in $token_{pay}$ tokens as a single payment, yet $interest$ is being converted into $token_{liq}$ tokens, using the $conv$ function. The result of this conversion is then automatically added back to the renting pool, thus ensuring *auto-compounding*. Important to note that auto-compounded funds are not immediately added to the renting pool reserves, but rather streamed there over the defined time (see 5.2.4).

5.2.4. *Renting Pool Reserves.* Formally, renting pool reserves are described by the following parameters (48).

$$\begin{aligned} \mathcal{R} &= \langle R_{fixed}, R_{used}, T_{allow}^{ref}, \alpha, T_{1/2}^{nonref}, streaming\rangle, \\ T_{allow}^{ref} &= \{T_i\}, \\ streaming &: T_{allow}^{ref} \cup \{*\} \to \{M_i\}, \\ M_i &= \langle target, fixed, t_{last}\rangle \end{aligned} \tag{48}$$

- Fixed part R_{fixed} of reserves represents amount of liquidity tokens that are provided by liquidity providers and extracted from non-refundable parts of loan interest payments;
- R_{used} is an amount of liquidity tokens currently locked in loans;
- T_{allow}^{ref} is a set of time periods available for refundable loans;

- α is a non-refundable portion of refundable loans;
- $T_{1/2}^{nonref}$ is a gap-halving period for the interest streaming of non-refundable loans;
- *streaming* is a mapping from extended set $T_{allow}^{ref} \cup \{*\}$ to *exponential streaming context* M_i, where $\{*\}$ represents a special *arbitrary period* that is used only in conjunction with non-refundable loans.

Exponential streaming context M_i describes a snapshot taken at t_{last} of exponential approaching to *target* starting at $fixed$ level. This snapshot is used to establish a time-driven value that represents the process of a continuous flow of interest payment from borrowers to the renting pool. Exponential streaming to be completely defined needs a corresponding $T_{1/2}$ (or equivalent λ). For every T_i we can define a corresponding $T_{1/2}$ via equations (49).

$$\alpha = \left(\frac{1}{2}\right)^{\frac{T_i}{T_{1/2}}}, \qquad T_{1/2} = \frac{T_i}{\log_{1/2}\alpha} \tag{49}$$

The only $T_{1/2}$ that cannot be calculated this way is in the case of non-refundable loans, where the time could be arbitrary. Thus, it should be provided explicitly as $T_{1/2}^{nonref}$.

$$\begin{aligned} R_{own}(\mathcal{R}, t) = R_{fixed} + \\ + s(fixed(M_*), target(M_*), T_{1/2}^{nonref}, t - t_{last}(M_*)) + \\ + \sum_{T_i \in T_{allow}^{ref}} s(fixed(M_i), target(M_i), \frac{T_i}{\log_{1/2}\alpha}, t - t_{last}(M_i)), \end{aligned} \tag{50}$$

where $M_i = streaming(T_i)$, $M_* = streaming(*)$ and s is a function defined at (9).

Here, using the additivity property of exponential curves with the same λ parameter, we avoid tracking the refunding streams of individual loans. Instead, we are using aggregated streams from all loans grouped by the same loan duration ($t_{end} - t_{start}$). This trick allows R_{own} to be calculated in $\mathcal{O}(|T_{allow}^{ref}|)$. It means T_{allow}^{ref} periods should be carefully chosen to reflect actual needs of borrowers. On the other hand, duration of non-refundable loans is not limited to a discrete set, but rather constrained by a range dur_{loan}^{nonref} (45).

5.2.5. *Bonding Curve.* A renting pool has its own bonding curve that defines a policy on how loan interest rate is calculated for all services, except that base prices are defined separately at the level of services. Bonding curve parameters r_{crit}, k and the function f_{bond} itself were discussed in detail in section (5.1.3).

$$\begin{aligned} bonding &= \langle f_{bond}, param_{bond} \rangle, \\ f_{bond} &: param_{bond} \times \mathcal{R} \times \mathbb{R}^{0+} \to \mathbb{R}^{0+}, \\ param_{bond} &= \langle r_{crit}, k \rangle \end{aligned} \tag{51}$$

5.3. **Renting Pool transactions.**

5.3.1. *Enterprise's side.* The life-cycle of a renting pool starts when the service provider *deploys an enterprise* by accessing the renting pool factory contract (main IQ Protocol contract). Renting pool creation involves specification of all renting pool specific parameters:

- $token_{liq}$ – liquidity token;
- γ_{gc} – percent of borrower's payment that will be used to incentivize social garbage collection of expired loans;
- r_{crit} – is a critical level of liquidity;
- k – is a slope of bonding curve;
- $conv$ – is a link to the conversion oracle/DEX.

After enterprise deployment, the service provider gets an exclusive role as an *enterprise owner*[6].The enterprise owner role allows for the change of almost any parameters specified on deployment. The only parameter that cannot be changed is $token_{liq}$. As well, deployment involves providing some meta- and human-readable information for usage in various integrations of IQ Protocol.

The enterprise owner has the following rights on the renting pool:

- Defining a new service Q_i;
- Enabling and disabling payment tokens $\mathcal{T}_{accepted}$;
- Collecting aggregated enterprise fee gathered from different Q_i to cover service running costs;
- Shutting down the enterprise.

Defining a new service Q_i includes the following parameters:

- $token_{base}$ – is the reference to the token in which service pricing is defined;
- $price_{base}$ – is a price (in $token_{base}$ tokens) of renting of one power token associated with Q_i service during one time unit;
- dur_{loan}^{nonref} – the maximum and minimum durations of non-refundable loans;
- fee_{serv} - the portion of interest payment used for covering service running associated costs;
- fee_{gc}^{min} - the minimum amount of $token_{base}$ tokens guaranteed to be included in social incentivization of returning expired loans;
- $T_{1/2}$ - proof of hold gap-halving period of the power token associated with Q_i service.

All parameters except $T_{1/2}$ could be changed by the enterprise owner after defining the service.

Service pricing. It is important to emphasize the difference between the three roles of tokens: $token_{liq}$, $token_{base}$, $token_{pay}$. The $token_{liq}$ is the only token that is stored in renting pool reserves, but it is not necessary to use it as a means of payment. Actual payment token $token_{pay}$ could be arbitrarily chosen by the borrower from $\mathcal{T}_{accepted}$. When payment for the same service can be accepted in different assets - we need some equivalent to express the price. This means that pricing should be defined using a single chosen asset $token_{base}$. For a healthy price structure - it is preferable to use a stablecoin as the $token_{base}$. Borrowers who are paying in stablecoins will not observe any changes in pricing even if $token_{liq}$ changes its price significantly. In case $token_{pay}$ is volatile and $token_{base}$ is stable, borrowers could experience volatility in price. However, the actual value of assets being paid

[6]Being an important role, the enterprise owner account should be protected by appropriate security techniques, like Multi-Sig, TimeLock etc.

will remain stable, as well, borrowers will be protected from the volatility during the period of the loan (there are no margin calls). These properties lead to a sustainable service consumption. In general cases, all three mentioned tokens can be different.

Accepted payment means. The enterprise owner can change the contents of the $\mathcal{T}_{accepted}$ set as long as the *conv* oracle DEX is supporting all of the necessary conversions (52).

(52)
$$\forall t_{pay} \in \mathcal{T}_{accepted}, \forall x \in \mathbb{R}: \{\langle token_{base}, t_{pay}, x\rangle, \langle t_{pay}, token_{liq}, x\rangle\} \subseteq \mathbf{dom}(conv)$$

Shutting down the enterprise. In emergency cases (e.g. token swap etc), the enterprise owner has a right to shut down the renting pool. This operation leads to an immediate unlocking of all renting pool reserves, even those which are locked in loans, so that liquidity providers can withdraw it. As a result, no loans can be borrowed anymore. However, existing loans can still be returned and fee_{gc} will still be claimable. Additionally, wrapping liquidity tokens to power tokens will become unavailable. However, unwrapping them will still be possible. These measures ensure that $token_{liq}$ will be fully extractable from the renting pool by the asset owners.

5.3.2. *Lender's side.* Lenders are supposed to be the original owners of $token_{liq}$ asset that was purchased during the ICO/IEO/IDO... or on the secondary market. The IQ Protocol assumes that only a fraction of the original asset owners are the consumers of the enterprise service. Since the $token_{liq}$ potentially represents a tokenized capacity/bandwidth of the enterprise - a lot of non-consuming holders appear. To improve their capital efficiency, they can decide to lend a part of their acquired capacity to users that are interested in the service consumption (borrowers). In case of multi-service renting pools, lenders just provide the liquidity for the entire enterprise without specifying which services will be used by borrowers. For those original $token_{liq}$ owners who want to actually consume services the IQ Protocol ensures an ability to convert (*wrap*) their $token_{liq}$ directly to power tokens in 1:1 equivalent of any available service Q_i.

Providing liquidity. When providing $token_{liq}$ to the renting pool, lenders are receiving iTokens in the form of an NFT. By its nature, iTokens are similar to deposit certificates, however they are not immutable and can evolve in time because of the following reasons:

- Implicit time-driven value, as with every borrow operation, the value of iToken starts to increase;
- Explicit lenders operations, such as increasing/decreasing liquidity, withdrawing interest, exiting liquidity (withdrawing liquidity and interest completely followed by the iToken burn).

Internally, iTokens are based on two parameters: shares ω and reference stake amount $amount_{stake}$. When lender identified by *addr* deposits x amount of $token_{liq}$ at time t into the renting pool with state $\mathcal{I} = \langle \Omega, stake, n\rangle$, reserves $\mathcal{R} = \langle R_{fixed}, \ldots\rangle$, the new iToken S_i is created with $amount_{stake} = x$ and shares ω are calculated according to the following equation (53).

(53)
$$\omega = \Omega \frac{x}{R_{own}(\mathcal{R}, t)},$$

where R_{own} is defined in (50).

The state of the renting pool is changed only after ω is calculated according to the following rules (alg. 1).

Algorithm 1: iToken issuance.

let $\omega = \Omega \frac{x}{R_{own}(\mathcal{R},t)}$;
$n_{stake} := n_{stake} + 1$;
let $id = \mathbf{hash}(n_{stake}||salt)$;
$stake[id] := \langle addr, \omega, x, t\rangle$;
$\Omega := \Omega + \omega$;
$R_{fixed} := R_{fixed} + x$;

Flushing streamed interest. The borrowing process increases pool reserves, using a time-driven strategy. It means, that actual available reserves are not only represented by R_{fixed} part of reserves, but also include several exponential streaming contexts that are processed independently and should be *flushed* into R_{fixed} when liquidity extraction operations are trying to determine whether a given amount of liquidity is extractable. Let's denote the flushing operation as **flush** and describe its semantics (alg. 2).

Algorithm 2: Flush operation.

for $T_i \in T^{ref}_{allow}$ **do**
  let $T^i_{1/2} = \frac{T_i}{\log_{1/2}\alpha}$;
  let $\langle fixed, target, t_{last}\rangle = streaming[T_i]$;
  let $\Delta R = s(fixed, target, T^i_{1/2}, t - t_{last}) - fixed$;
  $R_{fixed} := R_{fixed} + \Delta R$;
  $streaming[T_i] := \langle fixed - \Delta R, target - \Delta R, t\rangle$;
end
let $\langle fixed^*, target^*, t^*_{last}\rangle = streaming[*]$;
let $\Delta R^* =$
  $s(fixed^*, target^*, T^{nonref}_{1/2}, t - t^*_{last}) - fixed^*$;
$R_{fixed} := R_{fixed} + \Delta R^*$;
$streaming[*] := \langle fixed^* - \Delta R^*, target^* - \Delta R^*, t\rangle$;

For the sake of optimization, the flush operation can be triggered only when the withdrawn liquidity is more than R_{fixed}.

Interest withdrawal. Because of borrowing, the price of one share is constantly increasing, so full amount of $token_{liq}$ the lender can pretend to, is described by the equation (54).

(54)
$$amount_{full}(id, t) = \frac{\omega(stake[id])}{\Omega} R_{own}(\mathcal{R}, t)$$

By knowing $amount_{stake}$, it allows us to explicitly define *accrued interest*, as a time-driven value (55).

(55)
$$interest_{acc}(id, t) = amount_{full}(id, t) - amount_{stake}(stake[id])$$

Equation (55) allows us to define an operation of withdrawing accrued interest at time t for iToken id (alg. 3).

Algorithm 3: Accrued interest withdrawal.

let $\langle addr, \omega, amount_{stake}, t_{stake}\rangle = stake[id]$;
let $z = interest_{acc}(id, t)$;
if $R_{fixed} < z$ **then**
  flush R_{fixed};
end
if $R_{fixed} < z$ **then**
  error;
end
$R_{fixed} := R_{fixed} - z$;
let $\omega' = \Omega \frac{amount_{stake}}{R_{own}(\mathcal{R},t)}$;
$\Omega := \Omega - (\omega - \omega')$;
$stake[id] := \langle addr, \omega', amount_{stake}, t_{stake}\rangle$;
transfer$(token_{liq}, addr, z)$;

Liquidity exit. Lender can liquidate his/her stake by the following algorithm (alg. 4). All operations that decrease the provided liquidity are protected by the time-lock for one time unit: the lender cannot decrease or exit liquidity in the same transaction where he/she provided liquidity. That is the counter-measure against colluding borrowers and liquidity providers. Without this protection, the borrower can take a flash loan [16] in $token_{liq}$, provide it as the liquidity (affecting the price of the loan via bonding curve), take a cheaper loan and immediately exit the liquidity position with a subsequent return of the taken flash loan. The implemented counter-measure does not fully remove the possibility of colluding borrowers and lenders, yet it greatly increases the opportunity cost.

Algorithm 4: Liquidity exit.

let $\langle addr, \omega, amount_{stake}, t_{stake}\rangle = stake[id]$;
if $t_{stake} = t$ **then**
  error;
end
let $z = amount_{full}(id, t)$;
if $R_{fixed} < z$ **then**
  flush R_{fixed};
end
if $R_{fixed} < z$ **then**
  error;
end
$R_{fixed} := R_{fixed} - z$;
$\Omega := \Omega - amount_{stake}$;
burn $stake[id]$;
transfer$(token_{liq}, addr, z)$;

Modifying existing iToken deposit. The IQ Protocol allows lenders to modify iTokens, by adding liquidity to existing iTokens and removing liquidity from them partially (alg. 5). Let's consider a situation, when a lender wants to modify his/her $amount_{stake}$ of iToken id, by Δx $token_{liq}$ amount at time t. For the sake of simplicity, let's assume **transfer** operation with a negative amount as a transfer in the opposite direction (like $approve/transferFrom$ semantics in ERC20).

Algorithm 5: Modifying $amount_{stake}$.

let $\langle addr, \omega, amount_{stake}, t_{stake}\rangle = stake[id]$;
if $amount_{stake} + \Delta x < 0$ **then**
  error;
end
if $R_{fixed} + \Delta x < 0$ **then**
  flush R_{fixed};
end
if $R_{fixed} + \Delta x < 0$ **then**
  error;
end
let $\Delta\omega = \Omega \frac{\Delta x}{R_{own}(\mathcal{R},t)}$;
let $\omega' = \omega + \Delta\omega$;
$\Omega := \Omega + \Delta\omega$;
$R_{fixed} := R_{fixed} + \Delta x$;
$stake[id] := \langle addr, \omega', amount_{stake} + \Delta x, t'_{stake}\rangle$;
transfer$(token_{liq}, addr, -\Delta x)$;

5.3.3. *Borrower's side.* For borrowers, the following operations are available: estimate loan, borrow tokens, return loan and re-borrow tokens.

Estimate loan is a function that allows to calculate a price in $token_{pay}$ (provided by borrower) for borrowing $amount$ of power tokens for a certain period Δt for service Q_i (alg. 6).

Algorithm 6: Estimate loan.

function *estimate_loan*(Q_i, $token_{pay}$, $amount$, t_{start}, Δt, $isRef$) **is**
  let $\langle PT, token_{base}, price_{base}, loan, dur^{nonref}_{loan}, fee_{serv}\rangle = Q_i$;
  let dur;
  if $(isRef \wedge \Delta t \notin T^{ref}_{allow}) \vee (\neg isRef \wedge \Delta t \notin dur^{nonref}_{loan})$ **then**
    error;
  end
  let $price^{base}_{loan} = f_{bond}(R_{own}(\mathcal{R}, t_{start}), R_{used}, amount) \cdot amount \cdot \Delta t \cdot price_{base}$;
  let $price^{pay}_{loan} = conv(token_{base}, token_{pay}, price^{base}_{loan})$;
  let $fee^{pay}_{serv} = fee_{serv} \cdot price^{pay}_{loan}$;
  let $interest^{pay} = price^{pay}_{loan} - fee^{pay}_{serv}$;
  let $fee^{pay}_{gc} = max(conv(tokan_{base}, token_{pay}, fee^{min}_{gc}), \gamma_{gc} \cdot price^{pay}_{loan})$;
  return $\langle interest^{pay}, fee^{pay}_{serv}, fee^{pay}_{gc}\rangle$;
end

Borrow tokens is an operation that performs actual borrowing, while controlling the potential slippage (borrower can specify the maximum price of a loan based on prior loan estimation) (alg. 7).

Algorithm 7: Borrow.

function *borrow(Q_i, $token_{pay}$, amount, t_{start}, Δt, isRef, $amount^{pay}_{max}$)* **is**

- **let** $\langle PT, token_{base}, price_{base}, loan, dur^{nonref}_{loan}, fee_{serv}\rangle = Q_i$;
- **let** $\langle interest^{pay}, fee^{pay}_{serv}, fee^{pay}_{gc}\rangle =$ **estimate_loan**$(Q_i, token_{pay}, amount, t_{start}, \Delta t, isRef)$;
- **let** $payment = interest^{pay} + fee^{pay}_{serv} + fee^{pay}_{gc}$;
- **if** $payment > amount^{pay}_{max}$ **then**
  - **error**;
- **else**
  - **transfer**$(token_{pay}, addr, -payment)$;
- **end**
- **let** $loan^{pay} = interest^{pay} + fee^{pay}_{serv}$;
- **let** $loan^{liq} = conv(token_{pay}, token_{liq}, loan^{pay})$;
- **swap** $\langle loan^{pay}, token_{pay}\rangle$ **to** $\langle loan^{liq}, token_{liq}\rangle$;
- **let** $fee^{liq}_{serv} = loan^{liq} \cdot \frac{fee^{pay}_{serv}}{loan_{pay}}$;
- **let** $interest^{liq} = loan^{liq} - fee^{liq}_{serv}$;
- **transfer**$(token_{liq}, addr_{pool-owner}, fee^{liq}_{serv})$;
- **let** $dur,\ T_{1/2}$;
- **if** *isRef* **then**
  - $dur := \Delta t$;
  - $T_{1/2} := \frac{\Delta t}{\log_{1/2}\alpha}$;
- **else**
  - $dur := *$;
  - $T_{1/2} := T^{nonref}_{1/2}$;
- **end**
- **let** $\langle fixed, target, t_{last}\rangle = streaming[dur]$;
- **let** $fixed' = s(fixed, target, T_{1/2}, t_{start} - t_{last})$;
- $streaming[dur] := \langle fixed', target + interest^{liq}, t_{start}\rangle$;
- $n_{loans} := n_{loans} + 1$;
- **let** $id = \mathbf{hash}(n_{loans}||salt)$;
- $loan[id] := \langle addr, amount, t_{start}, t_{start} + \Delta t, token_{pay}, interest^{liq}, fee^{pay}_{gc}\rangle$;

end

Return loan is an operation to return borrowed loan amount back to the renting pool, while receiving a partial refund from the upfront paid interest in case if loan was taken in a refundable mode. Loan can be returned by the original borrower, so he/she can take the fee_{gc} reward back in case the return is made within allowed time period. When this period is over, then the right to return the expired loan to renting pool and grab the fee_{gc} reward is gradually passed to the renting pool owner and to everybody afterwards (alg. 8).

Algorithm 8: Return loan.

function *return_loan(id, t, caller)* **is**

- **let** $\langle addr, amount_{loan}, t_{start}, t_{end}, token_{pay}, interest, fee_{gc}\rangle = loan[id]$;
- **if** $isRef(id) \ \wedge\ t < t_{end} \ \wedge\ caller = addr$ **then**
  - **let** $\Delta t = t - t_{start}$;
  - **let** $dur = t_{end} - t_{start}$;
  - **let** $T_{1/2} = \frac{dur}{\log_{1/2}\alpha}$;
  - **let** $amount_{ref} = s(interest, 0, T_{1/2}, \Delta t)$;
  - **transfer**$(token_{liq}, addr, amount_{ref})$;
  - **let** $\langle fixed, target, t_{last}\rangle = streaming[dur]$;
  - **let** $fixed' = s(fixed, target, T_{1/2}, \Delta t)$;
  - $streaming[dur] := \langle fixed', target - amount_{ref}, t\rangle$;
- **end**
- **if** $t < t_{end} + T^{own}_{grace} \ \wedge\ caller\ != \ addr$ **then**
  - **error**;
- **end**
- **if** $t < t_{end} + T^{own}_{grace} + T^{pool-owner}_{grace} \ \wedge\ caller \notin \{addr, addr_{pool-owner}\}$ **then**
  - **error**;
- **end**
- **transfer**$(token_{pay}, caller, fee_{gc})$;

end

Re-borrow tokens is an operation that borrower can use to perform returning the original loan and taking another one atomically within single transaction without exposing liquidity to other potential borrowers. In case of re-borrow, the borrower still has to estimate the continuation of the loan as its price could change since original borrowing. Re-borrow operation is not available if current liquidity ratio is less than r_{crit}.

References

[1] *How we simulated and subsequently abandoned the MythX token.* URL: https://medium.com/atchai/how-we-simulated-and-subsequently-abandoned-the-mythx-token-472954520372.

[2] Aylin Aslan, Ahmet Sensoy, and Levent Akdeniz. "Determinants of ICO Success and Post-ICO Performance". In: Feb. 2021.

[3] Philipp Hacker and Chris Thomale. "Crypto-Securities Regulation: ICOs, Token Sales and Cryptocurrencies under EU Financial Law". In: *European Company and Financial Law Review* 15.4 (2018), pp. 645–696. DOI: doi:10.1515/ecfr-2018-0021. URL: https://doi.org/10.1515/ecfr-2018-0021.

[4] *Proposal for a REGULATION OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL on Markets in Crypto-assets, and amending Directive (EU) 2019/1937.* URL: https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=SWD:2020:0380:FIN:EN:PDF.

[5] Cameron Chell. *ICO's, Security tokens, Utility Tokens and AppCoins.* 2018. URL: https://www.linkedin.com/pulse/icos-security-tokens-utility-appcoins-cameron-chell/.

[6] Lin Cong and Yizhou Xiao. *Categories and Functions of Crypto-Tokens (June 29, 2020).* URL: https://ssrn.com/abstract=3814499.

[7] Paul Berg. "EIP-1620: ERC-1620 Money Streaming [DRAFT]". In: *Ethereum Improvement Proposals, no. 1620, November 2018. [Online serial].* URL: https://eips.ethereum.org/EIPS/eip-1620.

[8] Gavin Wood. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* 151 (2014), pp. 1–32.

[9] *Compound Finance.* URL: https://compound.finance/documents/Compound.Whitepaper.pdf.

[10] *Sablier Finance Streams.* URL: https://docs.sablier.finance/streams.

[11] Ulrich A. Müller. "Specially weighted moving averages with repeated application". In: *of the EMA operator, Olsen Research Institute Discussion Paper*. 1991.

[12] "Handbook of Measure Theory". In: ed. by E. PAP. Amsterdam: North-Holland, 2002, p. iv. ISBN: 978-0-444-50263-6. DOI: https://doi.org/10.1016/B978-0-444-50263-6.50042-7. URL: https://www.sciencedirect.com/science/article/pii/B9780444502636500427.

[13] Paresh Nakhe. "Dynamic Pricing in Competitive Markets". In: *CoRR* abs/1709.04960 (2017). arXiv: 1709.04960. URL: http://arxiv.org/abs/1709.04960.

[14] *Bonding Curve Definition.* URL: https://coinmarketcap.com/alexandria/glossary/bonding-curve.

[15] Guillermo Angeris, Alex Evans, and Tarun Chitra. *When does the tail wag the dog? Curvature and market making.* 2020. arXiv: 2012.08040 [q-fin.TR].

[16] Dabao Wang et al. "Towards understanding flash loan and its applications in defi ecosystem". In: *CoRR* abs/2010.12252 (2020). arXiv: 2010.12252. URL: https://arxiv.org/abs/2010.12252.