Analysis of Bancor Equations Supporting REX

Dan Larimer[1] and Khaled A. Al-Hassanieh[2]

[1]CTO, Block.one
[2]Senior Software Engineer, Block.one

Introduction

In this document, we describe the Bancor equations used in REX, and the reasoning behind setting the REX initial state and any possible constraints on the system. We first present the relevant REX pool balances. These balances are represented both by their C++ smart contract variable names and by the mathematical variables used in the presented equations. If not shown explicitly, the unit of all balances, paid fees and staked resources is the blockchain core token SYS. The balances are

- *total_unlent* represents the SYS balance that is available for renting. In the following we use u to represent this balance, i.e., $u = \mathit{total\_unlent}$.

- *total_lent* represents the total rented SYS balance. At any point in time, *total_lent* is the sum of tokens staked in all currently open loans. In the following we use $l = \mathit{total\_lent}$.

- *total_rent* is a virtual balance. The initial value of this balance must be strictly positive as discussed below. The balances *total_rent* and *total_unlent* are the two connectors of the Bancor algorithm which determines CPU and Network renting prices. In the following we use $f = \mathit{total\_rent}$.

For a detailed description of the REX smart contact, see Ref [1].

REX Loan Calculations

Upon renting CPU or Network resources, the amount staked to those resources for 30 days is calculated as a function of the loan fee, Δf, and the REX pool balances $u = \textit{total\_unlent}$ and $f = \textit{total\_rent}$, using Bancor equation. For a given loan with id i, the equation is

$$\Delta u^{(i)} = \Delta f^{(i)} \frac{u}{f + \Delta f^{(i)}}. \tag{1}$$

For example, if at a given point in time, $u = 5 \times 10^7$, and $f = 3 \times 10^4$, a loan fee $\Delta f^{(i)} = 1$ results in renting $\Delta u^{(i)} = 1666.6111$ SYS worth of resources. That is, the renting cost rate for this loan is $\Delta f^{(i)}/\Delta u^{(i)} \approx 0.06\%$. In general, the REX pool balances are large compared to the fee of a given loan, then the renting cost rate can be estimated as

$$r \approx f/u. \tag{2}$$

When the loan described above is created, the REX pool balances are updated as follows: $u \to u - \Delta u^{(i)}$, $l \to l + \Delta u^{(i)}$, $f \to f + \Delta f^{(i)}$. In other words, the staked amount $\Delta u^{(i)}$ is moved from u to l, i.e., from *total_unlent* to *total_lent*. In addition, the paid fee is added to *total_unlent*. The overall set of updates is

$$\begin{array}{rcl} u & \to & u - \Delta u^{(i)} + \Delta f^{(i)}, \\ l & \to & l + \Delta u^{(i)}, \\ f & \to & f + \Delta f^{(i)}. \end{array}$$

Note that f is a virtual balance and there is no double spending by adding $\Delta f^{(i)}$ to both u and f.

Initializing REX Pool

Initially, the REX pool is empty ($u = l = 0$). As lenders buy REX (lend SYS tokens), u increases. On the other hand, the balance f is virtual and needs to be initialized to some value f_0. It is important to note that f_0 must not be zero; otherwise, the first loan will deplete the entire u balance no matter how small the paid fee is. This can be easily verified by setting $f = 0$ in Eq 1 which results in $\Delta u = u$ for any $\Delta f > 0$. Seeing that we must have $f_0 > 0$, the next step is to decide a practical value of f_0. The REX pool balance u is expected to reach tens of millions of SYS tokens rather quickly. We will use the estimate $u_0 = 2 \times 10^7$ as a reference value. A small f_0 causes a problem similar to the one caused by $f_0 = 0$. For example, if $f_0 = 100$, a payment $\Delta f = 100$ gives a rented stake of $\Delta u = 1 \times 10^7$, which is half of the entire pool. The same can be repeated and most of the pool can be rented using only a small sum of fee payments. Following the first few loans, renting cost increases rapidly and becomes too high.

On the other hand, setting f_0 to a large value would lead to a prohibitively high renting cost. By setting a target initial renting cost rate $r_0 \approx 0.1\%$, and using the u_0 reference balance, Eq 2 gives $f_0 = 2 \times 10^4$, which is the initial value we choose for *total_rent*.

Loan Expiration

When loan i expires, the corresponding rented resources, $\Delta u^{(i)}$, are released, i.e., moved from *total_lent* back to *total_unlent*. The balance f is updated by subtracting the output of the inverse equation

$$\Delta f'^{(i)} = \Delta u^{(i)} \frac{f'}{u' + \Delta u^{(i)}}, \tag{3}$$

where $\Delta u^{(i)}$ was calculated using Eq 1, and u' and f' are the values at loan expiration. Since these values are in general different from the values at loan creation, $f' \neq f$ and $u' \neq u$, we have $\Delta f'^{(i)} \neq \Delta f^{(i)}$. That is, the output of Eq 3 is different from the fee paid at loan creation. To summarize, the updates are

$$\begin{aligned} u' &\rightarrow u' + \Delta u^{(i)}, \\ l' &\rightarrow l' - \Delta u^{(i)}, \\ f' &\rightarrow f' - \Delta f'^{(i)}. \end{aligned} \tag{4}$$

Looking at Eq 3, we notice that if, at the time of expiration, *total_unlent* happens to be zero ($u' = 0$), the equation gives $\Delta f'^{(i)} = f'$. And following the update given by Eq 4, we get $f' = 0$ after the loan expires. As described above, this leaves the market in an unstable state. One scenario that can lead to this state is as follows: while there is at least one outstanding loan, one or more REX owners may sell enough REX to cause *total_unlent* to drop to $u' = 0$. Following that, one or more loans can expire resulting in $f' = 0$.

In order to prevent the system from reaching that state, we impose a dynamic lower bound on u which we describe in the following section.

Unlent Balance Lower Bound

Let u_{lb} be the dynamic lower bound of u, which means that at any point in time we have $u \geq u_{lb}$. We must define u_{lb} such that $u_{lb} > 0$ as long as there are outstanding loans, and $u_{lb} = 0$ when all loans have expired. The second condition allows REX owners to sell all their REX. Setting u_{lb} to be a fraction of l, i.e., $u_{lb} = \alpha \times l$, where $0 < \alpha < 1$, satisfies both requirements. In addition, we want a reasonably low u_{lb} so that it does not routinely cause selling orders to be queued and renting actions to fail. We set $\alpha = 0.2$, i.e., $u_{lb} = 0.2 \times l$.

Note that we chose to calculate u_{lb} as a function of l instead of f for two reasons. Fist, u is expected to be of a different order of magnitude than f which makes the comparison impractical, and second, the value of f cannot be used to determine whether there are outstanding loans.

Adjusting REX Pool Virtual Balance

We provide a backup solution that can be invoked in case REX initial condition is out of balance. This can happen, for example, if after a period of time, *total_unlent* remains well below the reference value of $u_0 = 2 \times 10^7$ described above. It means that the initial renting cost rate is well above target value $r_0 \approx 0.1\%$, or the target rate determined by similar resource renting markets.

The action *setrex* allows producers to set the balance f to a predetermined value calculated using Eq 2 as $f_0 \approx r_0 \times u$, where u is the current value of *total_unlent* and r_0 is the target renting cost rate.

Derivation of the Equations

Bancor protocol [2] allows for instant liquidity by connecting a currency reserve to a smart token. It defines the fractional reserve ratio as

$$F = \frac{R}{SP},$$

where R is the current value of the currency reserve, S is the smart token current supply, and P is the current token price relative to the reserve currency. The protocol posits that F is always constant and is set to a predetermined value which dictates the price behavior as a function of supply.

One of the results of the protocol is an equation that determines the amount to be paid in return for a given number of tokens:

$$\Delta R = R_0 \left[\left(1 + \frac{\Delta S}{S_0} \right)^{\frac{1}{F}} - 1 \right], \tag{5}$$

where R_0 is the initial reserve value, S_0 is the initial smart token supply, and ΔS is the number of issued tokens.

The inverse equation,

$$\Delta S = S_0 \left[\left(1 + \frac{\Delta R}{R_0} \right)^{F} - 1 \right], \tag{6}$$

determines the number of smart tokens issued in return for a given payment. After the tokens are issued, the supply is updated to $S = S_0 + \Delta S$ and the reserve to $R = R_0 + \Delta R$.

Now consider a smart token that is connected to two reserves $R^{(1)}$ and $R^{(2)}$, and assume that the fractional reserve ratio of the smart token is the same for both reserves. A payment $\Delta R^{(1)}$ results in ΔS issued tokens given by Eq 6 applied to $R^{(1)}$:

$$\Delta S = S_0 \left[\left(1 + \frac{\Delta R^{(1)}}{R_0^{(1)}} \right)^{F} - 1 \right] \implies \frac{S_0 + \Delta S}{S_0} = \left(\frac{R_0^{(1)} + \Delta R^{(1)}}{R_0^{(1)}} \right)^{F}. \tag{7}$$

If these tokens are then sold (equivalent to adding $-\Delta S$ to the smart token supply) in exchange for the second reserve currency, we obtain

$$\Delta R^{(2)} = R_0^{(2)} \left[\left(1 - \frac{\Delta S}{S} \right)^{\frac{1}{F}} - 1 \right] = R_0^{(2)} \left[\left(\frac{S_0}{S_0 + \Delta S} \right)^{\frac{1}{F}} - 1 \right]. \tag{8}$$

Replacing Eq 7 in Eq 8 results in

$$\Delta R^{(2)} = -R_0^{(2)} \frac{\Delta R^{(1)}}{R_0^{(1)} + \Delta R^{(1)}}.$$

Note that $\Delta R^{(2)}$ and $\Delta R^{(1)}$ have opposite signs. In REX equations shown above, the two reserves are $f \equiv R^{(1)}$ and $u \equiv R^{(2)}$.

References

[1] REX Implementation. `https://github.com/EOSIO/eosio.contracts/issues/117`

[2] Eyal Hertzog, Guy Benartzi, and Galia Benartzi. Bancor Protocol White Paper. `https://storage.googleapis.com/website-bancor/2018/04/01ba8253-bancor_protocol_whitepaper_en.pdf`