Flow Control Currency

Team Flocc

Eidgenössische Technische Hochschule Zürich (ETHZ)

Magnus Wuttke, wuttkem@student.ethz.ch
Colin Rix, crix@student.ethz.ch
Ying-Kai Dang, ydang@student.ethz.ch
Gregory Inauen, ginauen@student.ethz.ch
Alessandro Buser, buseral@student.ethz.ch
Fabian Mächler, mafabian@student.ethz.ch
Kapgen Tun, tkapgen@student.ethz.ch

ETH Zürich

April 30, 2018

All contributors contributed equally to this paper. The code was mainly written by Ying-Kai Dang, Alessandro Buser and Fabian Mächler.

Contents

I. Introduction

Lately, cryptocurrencies and the blockchain technology itself brought remarkable changes to the way we, as users, generally perceive trusted parties during interactions in any given system. For example, speaking of monetary transactions, the trusted parties in this case could be banks that take care of their clients' money. Here, the relation between the monetary institution and the client is based on a substantial amount of trust from the client side.

By implementing a technologic system, based on a peer to peer network, all of the intermediary units (e.g. institutions) are eliminated. Therefore, the end-users obligation to trust intermediaries is shifted to complete confidence in technology, which, ideally, is almost impossible to manipulate. Apart from these changes, cryptocurrencies have not yet managed to fundamentally revolutionize the underlying financial system. Thus they inherited several of the major problems we are seeing and experiencing currently in our everyday life in the real world. Our belief is that blockchain technology has the disruptive potential and the power to allow the creation of a self-regulating system, that in itself is more sustainable, stable and trustworthy than the already existing systems.

II. Problems

i. Monopolization of value

One of the main problems of current systems is the vulnerability to monetary monopolies. When single parties hold a significantly high amount of a currency, they gain massive influence over the corresponding system and therefore also on its economical behaviour and development. By holding back ("stockpiling") the currency they own, these parties obtain the ability to dictate its value by provoking artificial deflation and inflation. This way, they can achieve a further increase in their power over the system. As a consequence of this manipulation, the monetary structure is destabilized, having a negative impact on everybody with less influence. Because this disrupts the ordinary everyday usage of the currency as a means of payment. This major flaw leads to systems heavily favouring parties who possess significantly higher capitals than the average person.

ii. Currency stability

As discussed before, the value of any given currency can be manipulated to favor the interests of parties holding important amounts of it. Additionally, the value of a currency can be fluctuating because of speculation and market principles depending on the underlying financial concept. This results in a reduction of usability of the currency for everyday applications, e.g. payment. We also notice a severe impact from political and real world events on the stability of currencies, resulting from a lack of trust in this very stability. This cyclic dependency is increased even more by untrustworthy centralized currency controlling institutions.

iii. Currency flow and Economy

Currently existing real world systems use inflation as a regulatory mechanism to keep currency "flowing" and to punish parties holding currency instead of spending it. The problem with inflation is that it is punishing everybody for holding currency. Although there exist several scenarios in which parties are required to hold and accumulate larger amounts of currency, such as risk capitals, retirement provisions, large projects, real-estate purchases, etc. Moreover, inflation is permanently decreasing the value of a currency and comes with the need for a centralized control unit creating the currency and managing it. A different approach, used by systems such as Bitcoin and most other cryptocurrencies (commonly called Altcoins), consists of limiting the total amount of available currency and leads as a consequence in the elimination of inflation. This is per se not a bad thing, but these systems lack reasons for people to keep currency circulating instead of holding it. Unfortunately, this can be limiting the economic potential and favoring the accumulation of big capitals and monopolies in a similar manner as was described before.

III. Approach

i. Flow control currency

In this section we make efforts to explain the key points of our attempt to solve the previously discussed problems. In order to properly describe the concepts that we developed, the explanations here will eventually go a bit deeper into the technical details.

The core idea of our team consists of extending, rather than replacing, existing systems in order to improve the situation. We chose a system without inflation, this means that a constant amount of our primary currency is provided, functioning as the base of the system. We extend this primary currency by the introduction of a secondary currency, denoted as flow control currency (FLOCC), responsible for controlling and stabilizing the flow of the primary currency inside the system, influencing it through its main mechanism. The flow control currency is achieving to do so by rewarding people for spending and using primary currency.

Subsequently, we will use "FLOCC" when referring to the concept of a flow control currency and "PC" when referring to the concept of a primary currency. *pc* and *flocc* will be used as placeholders to refer to actual currencies acting as PC, respectively FLOCC, and to refer to the base units of these currencies. For convenience, we do not distinguish between the name of the base unit and the corresponding currencies. We separate the two currencies conceptually by assigning every *pc* and every transaction involving *pc* to what we denote as the primary economic space (PES) and analogous every *flocc* and every transaction involving *flocc* to the secondary economic space (SES).

The main mechanism acts as follows: Person A is spending 1 *pc* and receives Γ *flocc*, which have been created by the system. Γ is therefore the amount of *flocc* received per pc spent, called Generation Rate. By providing an incentive to spend *pc*, the mechanism is improving the flow inside the PES, while reducing the profit one can gain by holding large amounts of *pc*.

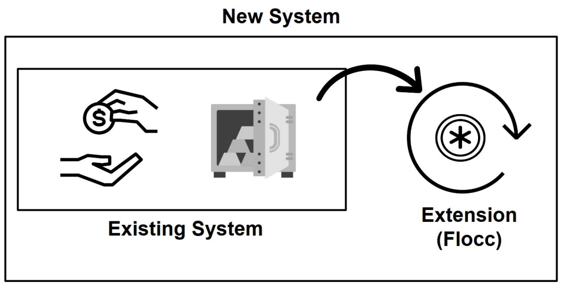

Figure 1: *The PES is extended by the SES*

ii. Properties

We determined several intrinsic properties of FLOCC that are necessary for the implementation, i.e. improving its stability and avoiding the reproduction of issues known from the system being extended by a FLOCC.

- **Temporality:** In order to avoid the possible monopolization of a FLOCC and increase the currency flow inside the SES, a FLOCC has to be temporary. This means that when owning FLOCC, it has to be spent within a time interval T, otherwise it will be removed from its current owner. We denote T as the Holding Time. T determines that one can only hold a certain amount of *flocc* earned at time t_1 up until time $t_2 = t_1 + T$. At time t_2 the *flocc* is lost if it was not spent in the meantime.
- **Limited total amount:** We have chosen a system without inflation as the base system being expanded (PES inflation free), so we have to make sure we are not intro-

ducing inflation within the SES. In order to do so, we define M as the maximum amount of *flocc* allowed in the SES.

- **Limited personal amount:** One issue a FLOCC is attempting to solve is single parties or people gaining too much influence over the system as a whole (both PES and SES). We denote the amount of *pc* currently owned by person A by P_A, the amount of *flocc* held by this person by F_A and the total amount of *pc* by N. We further define LA as the maximum amount of *flocc* the system allows a Person A to hold with $LA = f(M, N, P_A)$ for a monotonically decreasing function f . The provided solution makes sure that parties holding significant amounts of PC and therefore having a wide influence on the PES, will have little or no influence on the SES. This represents a self-regulating mechanism, enabling the system to stabilize itself without any exterior interaction. If Person A wants to hold more *flocc* and $F_A = L_A$, A has to spend *pc* to increase L_A. This property acts as an incentive for people to spend *pc* if they wish to own more *flocc* and increase their influence on the SES.
- **Reattribution:** By looking at the last two properties, we see that their definition created a problem regarding the generation mechanism of a FLOCC. If all units of *flocc* have been distributed, i.e. $\sum_{A \in Persons} F_A = M$, we have to figure out how someone can be rewarded for spending *pc*. The solution we propose is a mechanism called Reattribution. Reattribution is the transfer of the appropriate amount of *flocc* from the person holding the *flocc* with the least recent timestamp to the person that spent *pc*. The system keeps track of every amount of *flocc*, who it belongs to and when it has been obtained (by either generation or transfer). This tracking enables the system to know who has been holding *flocc* for the longest time without spending it. This can be understood as the system attributing *flocc* to someone spending *pc*, the intention being the *flocc* on his part being spent. If this does not happen, the system reattributes the *flocc* to someone else, so that this person can spend it and benefit from it. The system wants to improve the *flocc* circulation and is therefore fighting the build-up of monopolies.

iii. The Control and Voting Unit

We already presented several reasons for people to spend *flocc*, but now we also need reasons why users would even want to hold amounts of *flocc*. By providing incentives both for spending and holding *flocc*, the system is able to regulate itself. This is why the Control and Voting Unit (CVU) is introduced, providing a mechanism to influence the Generation Rate Γ and the Holding Time T. The CVU is implemented using a DAO (Decentralized Autonomous Organization) and it accepts votes from FLOCC users to decide the settings of Γ and T. In order to do this, the corresponding weighted averages of the votes are used, where the amount of *flocc* F_A is acting as the stake (voting weight) of a Person A. Here again, the system provides a self-regulating mechanism, through only allowing people to vote for Γ and T in previously defined, but dynamic, ranges. These ranges are depending on the *pc* flow (amount and frequency) inside the PES. As a result, the ranges also depend on the amounts and frequency of *flocc* Generation and the frequency of Reattribution.

Higher traffic in the PES

- **Generation Rate:** Smaller range located at lower values. A lot of *pc* is circulating and therefore the system stabilizes itself by reducing the incentive for spending *pc* and preventing to much Reattribution taking place.
- **Holding Time:** Wider range located at higher values. A lot of *flocc* is generated and reattributed and therefore the system can allow people to hold *flocc* for a longer time. The gain of influence over the sys-

tem is compensated by the increased risk of losing *flocc* through Reattribution.

Lower traffic in the PES

- **Generation Rate:** Wider range located at higher values. The system seeks to improve the *pc* flow and is therefore rewarding the spending of *pc* by higher gains of *flocc*.
- **Holding Time:** Smaller range located at lower values. Only few amounts of *flocc* are generated and reattributed, therefore the system is limiting the Holding Time to compensate the decreasing risk of losing *flocc* through Reattribution and by consequence limiting the influence of people on the SES. These adaptations further avoid *flocc* monopolies and increase the traffic inside the SES.

By enabling the community of the FLOCC holders to influence the Generation Rate Γ, the system implicitly enables people to modify the value of the FLOCC relative to the value of the underlying PC. Therefore, a FLOCC has the potential to be relatively stable, as fluctuations of the PC can be compensated by adapting the Generation Rate, which is acting as a buffer.

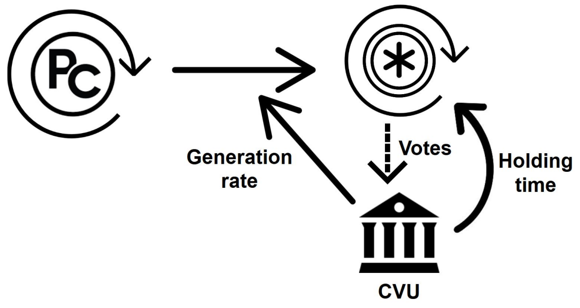

Figure 2: *Main cycle and dependencies of an extended system*

IV. Implementation

i. Theoretical Requirements

- **ERC20:** We used the ERC20 Token as a model for our Flocc-contract. The ERC20 Token has the following properties:
  - A uint called `totalSupply` represents the total sum of all balances.
  - Another basic property is the `balanceOf` uint. It holds a uint to each address which represents the current value of the wallet.
  - The `transfer()` function sends a set amount of tokens from the message sender to a specified recipient.
  - Although the transfer function can be used to send tokens from one address to another, when another smart contract requires a payment to execute a function, the smart contract does not have the ability to see which address sent the funds at the time, thus not knowing which user to approve. `transferFrom()`, along with the `approve()` function, allow third parties to spend tokens on their behalf. The token holder gives the third party, the smart contract, the approval to spend a specified amount of tokens, called `allowance`. Once approved, the third party can use the `transferFrom()` function to send the allowed amount of tokens to another address.

- **FLOCC-System:** To make the FLOCC-system work, we need several properties in addition. We divided the whole system into 2 modules (as also shown in Figure 2). The CVU as the module for controlling and voting over the Holding Time and Generation Rate, the second module being the FLOCC contract itself, which implements all the properties from the ERC20 Token plus a few additional in order to keep the properties as presented above.

In the following section we are going more deeply into each property to give the reader a better understanding of our implementation. Since we want a Token to expire after a certain amount of time, we need to let it expire it at the right moment. Letting it expire in this context means that the token gets destroyed if either the market is full and a user gets a new FLOCC-Token or in general the Holding Time is over.

ii. Implementation

i. DAO

The DAO contract is the owner of the FLOCC contract and acts as the aforementioned Control and Voting Unit. A user needs to pay 1 Wei in order to vote and can then set their proposed Holding Time and Generation Rate. If the user has not voted yet, the total amount of addresses that have already voted increases and the address of the user gets marked. The amount of addresses that already voted is needed to calculate the weighted average of the Holding Time and Generation Rate. Both the user-proposed Generation Rate and Holding Time are then stored in an array. If the pool is exceeded, the function `updatePool()` is called.

In `updatePool()` the weight of each vote is calculated, where the current amount of tokens held by the address determines the weight. The more tokens, the higher the weight. All proposed Holding Times and Generation Rates are then summed up, scaled accordingly to the weight of the proposer. Eventually, the new Holding Time and Generation Rate is updated and all values for a new proposition are reset. This is all executed with one call, the triggering of the pool, which also means that the gas price for triggering the pool is much higher than voting at any other given time. Therefore, the user who eventually called the `updatePool()` function also needs to pay those costs out of his own pocket. To compensate, that user will receive the whole pool, which is higher than the gas price.

ii. FLOCC

Each transaction is stored in a mapping `transactionLedger` to keep track of all transaction, but most notably the oldest transaction whose time has not exceeded the set Holding Time, and the latest transaction. There are also two pointers, `minPointer` and `maxPointer`, which both point to the first transaction at the initialization. With each additional transaction, the `maxPointer` moves to the latest transaction.

With each call that checks or updates the balance of an address a function `checkTime()` is called. That function checks whether the oldest transaction in the `transactionLedger` has expired or not, i.e. exceeds the Holding Time, and updates the `minPointer` to the oldest transaction which has not expired yet. Once the function finds such an expired transaction, amount of *flocc* of the transaction sender is deducted by the amount of *flocc* of the transaction.

For each spent *pc* the user also gets *flocc*. This is created with `mintToken()`, which can only be called by the owner, that is the DAO. `mintToken()` has two cases:

- *M, the maximum amount of flocc in the SES, has already been reached:* Here the reattribution property of FLOCC comes into play. As all *flocc* are already on the market, no new token can be created. As a result, the oldest transaction that hasn't expired yet, meaning the transaction with the `minPointer`, here called T_{m1}, will be chosen for reattribution. Here we can distinguish between two cases again: The transaction T_{m1} has enough *flocc* for reattribution and the transaction does not have enough.
  In the first case, the amount of reattributed *flocc* is simply reattributed from the holder of the transaction T_{m1} to the new transaction holder.
  In the second case, the chosen transaction T_{m1} did not transact enough *flocc* to cover the newly attributed *flocc*. This means the

remaining *flocc* that are not covered by T_{m1} will need to be taken from the next transaction T_{m2}. This will go on until the whole amount of *flocc* has been covered.

- *M has not been reached:* New tokens will simply newly created and the amount of *flocc* currently in circulation is updated. If the *flocc*-cap is reached with that one transaction, meaning the creation of new *flocc* leads to an excess of M, reattribution will occur in the same manner as described in the previous point.

Our implementation does not take into account yet that the holder of the to be reattributed *flocc* has already spent those expired *flocc*, making it possible for the holder to have a negative balance. To understand the cases more clearly, please consider following example:

The maximum amount of *flocc* is set to 30 with a Holding Time of 200.

Person A generates 5 *flocc* as the very first transaction in the system. Both the `minPointer` and the `maxPointer` point to that transaction. Then Person B generates 10 *flocc* immediately after that and `maxPointer` will be updated to that transaction. Person C will then generate 8 *flocc* and the `maxPointer` is updated, afterwards Person A generates another 4 *flocc*. Eventually, the `transactionLedger` looks like following:

`transactionLedger`

Person	*flocc*	Timestamp	Pointer
A	5	0	`minPointer`
B	10	80	
C	8	140	
A	4	180	`maxPointer`

Person	Balance
A	9
B	10
C	8

Total *flocc* in circulation: 27

Now a fifth transaction occurs. Person C generates 9 more *flocc*, though as that will exceed M, part of the generated 9 *flocc* need to be reattributed from the transaction with the `minPointer`. 3 *flocc* can be still generated as the cap hasn't been reached yet, but because the `minPointer` transaction has only an amount of 5 *flocc*, the remaining one *flocc* needs to be reattributed from the next transaction. Eventually, the `transactionLedger` and the balances will look like following:

`transactionLedger`

Person	*flocc*	Timestamp	Pointer
A	0	0	
B	9	80	`minPointer`
C	8	140	
A	4	180	
C	9	200	`maxPointer`

Person	Balance
A	4
B	9
C	17

Total *flocc* in circulation: 30

iii. Owner

As owner of the Flocc contract, the CVU or DAO is the only one who is able to make internal changes. The deployer has to declare the DAO as owner of the Flocc contract at the initialization, otherwise the automation of generating *flocc* cannot be done. Furthermore, only the owner can mint tokens, execute transfers from one address to another, update Holding Time and Generation Rate and set the cap of *flocc*.

V. Inspiration

As we had noticed all the problems which were already described in this report beforehand, we were thinking on how we could stabilize the cryptocurrency market without a powerful institution like a central bank. We wanted to implement a completely decentralized system, so that the affected cryptocurrencies could remain decentralized and would not have to be influenced by any centralized institution in order to become stable. Our goal is to stabilize the market as a whole by motivating people to regularly spend their cryptocurrencies. Through achieving this, a constant flow of currency is obtained in the crypto market system.

After a few hours of developing our ideas, we decided that we wanted to implement some kind of a reward system, because people tend to perform actions when they get rewarded for doing it. Users who spend their cryptocurrencies would receive benefits for doing that and therefore, using cryptos as a means of payment (instead of holding them) would become an advantage. We decided that the reward should be a token. However, the problem of how to generate a constant flow was still remaining. Now people would probably tend to spend large amounts of cryptocurrency abruptly to receive a lot of tokens. Later, we figured out that we could implement a rate for the number of tokens generated per crypto transaction (i.e. Generation Rate). We noticed that with this idea we could also support the users who do not have high balances of currencies as these people are strongly dependent on the actions people with significant amounts of cryptocurrencies make. Those rich people or institutions acquire monopolies and thus have the power to influence the market price of the respecting cryptocurrency, by either holding their coins back or flooding the market with them. To give more power to the less influential part of the users, we came up with the idea of a voting system. Everyone who owns *flocc* can vote for a so-called Holding Time (T) and for a Generation Rate (Γ). But, the amount of *flocc* that one is capable to own is limited by the relative amount of *pc* the person possesses. As a result, the more other currencies one has, the less flocc one can obtain. Therefore, the people which high balances in other cryptocurrencies can not get too much influence in the votes of *flocc* (so it is impossible to influence the regular system and the FLOCC system at the same time).

The implementation of a limited Holding Time is needed to prevent that a single individual can get too powerful in the FLOCC system. If one person has large amounts of *flocc*, he/she can take much influence for one voting. But because of the limited Holding Time, only someone who is constantly spending a lot of *pc* can keep his/her amount of *flocc* (and therefore influence) constant. In order to achieve that, a very large amount of *pc* is needed, but this again limits the total amount of *flocc* one can hold. These counter-acting mechanisms are supposed to bring a self-regulating property into the system.

A big question remained: Who will pay for all this? The votes themselves and the updates of the votes of the users need to be paid by someone as it is stored on the blockchain. We decided that the whole community will pay for it together. After a long brain storming session, we decided that every update of a vote which is done will cost 1 Wei more than the actual update would cost. This 1 Wei will be send to our decentralized autonomous organization (DAO), which we used to implement our voting system. If the pool in the DAO is filled with enough gas to pay for the overall voting, the vote will be initialized. Per our design the DAO cannot pay the transaction costs itself, therefore the initialization must be triggered by a user. As those cost can be rather high, we thought of compensating the user who triggered the pool by transferring him all Ether that is in the pool at the time of triggering. This also encourages users to vote, as inevitably there is more in the pool than is costs to trigger the pool. One could say it acts as a lottery system which also has the effect of bringing more people to the ballot, thus increasing the flow in the system.

VI. Future Goals and Visions

The concept presented in this report and the underlying mechanisms could of course be used to create a completely new and independent structure, which could act as a decentralized and worldwide monetary system. However, we believe the true potential of a FLOCC lies

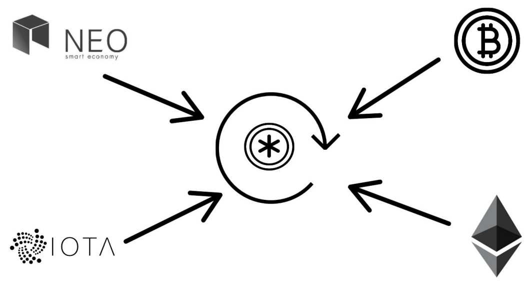

Figure 3: *A shared FLOCC extending and bridging several PCs*

in its capability of extending already existing systems, since it can be built on top of them and absolutely no changes of the underlying PC are required in order for the mechanism to work.

In fact, every current cryptocurrency could be extended by a FLOCC and would then benefit from the positive and self-regulating effects described above. Creating a separate FLOCC for every cryptocurrency would stabilize the currency itself, but not imperatively the value of currencies relative to each other. Our vision is to create a shared FLOCC, expanding and bridging all cryptocurrencies together and therefore bringing all of its positive effects to every single one of them. While, on top of this, assuring a relative stability of the underlying cryptocurrencies between each other. This system would use the cryptocurrencies as PCs and enable people to hold and save money if needed, while using the FLOCC as a daily currency, easily accessible and stabilizing the whole system by its existence and its circulation.

As a final note, we highlight the potential of a FLOCC for being used as the stake in a system, based on a proof of stake consensus mechanism. A classical proof of stake system, as presently used in cryptocurrencies, favors parties holding significant amounts of PC and since PCs are not temporary, the influence of these parties is rather increasing than decreasing. A FLOCC on the other hand is temporary and thus monopolies cannot be maintained for a long time. As a consequence, using a FLOCC for a proof of stake consensus would result in a more uniform distribution.

All of this being highly hypothetical, a lot of research has to be done and several technical and security issues have to be solved. As examples of what still has to be figured out we list *Cyclic Generation*, consisting of multiple people (possibly even a single person with multiple accounts) moving *pc* back and forth and by this generating arbitrary large amounts of *flocc*, how to ensure that the user who triggers the pool holds enough *pc* to pay the gas before getting the reward, and *Cyclic Holding*, consisting of multiple people (or again a single person with multiple accounts) moving *flocc* back and forth and by this never exceeding the Holding Time and never being in danger of holding the *flocc* with the least recent time stamp. As a consequence, they could potentially hold their *flocc* for an arbitrary amount of time without having to spend it. Further self-regulation mechanisms have to be conceived and their implications taken into account in order to avoid this kind of problems and to increase the stability of the FLOCC-extended system.

VII. Additional Information

i. Code

The software code which is part of this report is open source and available at `https://github.com/ETHBiots2018/Flocc`

This project report was written as part of the spring 2018 course "Blockchain And the Internet of Things (851-0591-01L)" run by M. Dapp, S. Klauser, and D. Helbing.