Delegated Range Voting with Tokens and Shit

Fabrizio Romano Genovese, Jelle Herold
Swarm Team
fabrizio@swarm.com jelle@swarm.com

After some brainstorming sessions in which the main ideas have been laid down, we start drafting how a hypothetical Liquid Democracy implementation for Swarm could look in technical terms. We will use a typed approach, that should in principle be compatible with the concepts behind StateBox.

1 Why voting with tokens as it is is not a good idea

The standard voting with tokens works like this: Everyone obtains some voting tokens in some way (for instance, one receives an amount of voting tokens equal to the tokens owned). These tokens then get sent to an address that fundamentally means "yes" or "no". Tokens sent in any address are then counted and a decision is made. This has three fundamental problems:

1. This voting procedure is basically majority voting, that is super flawed.
2. In the very same moment someone votes, the tokens are transferred to the other address and can be publicly seen. This means that a sufficiently skilled user is able to see in real time who is winning the vote. This is very bad because gives away data useful for any kind of manipulation.
3. If a voter sends his "real" tokens to someone else, he retains his voting tokens. This means that the other person receiving the tokens has no voting power whatsoever. On the other hand, the person retaining the voting tokens has less or no stake at all.

   For instance, someone with a lot of real tokens could sell them all, and then use the voting tokens he retained to vote for a very bad decision for the platform. At this point the tokens he just sold will be depreciated because of this bad decision and the person will be able to buy them back at a fraction of the price, making a profit. This means that a very powerful user on the platform can exploit the democratic infrastructure to make direct profits.
4. Delegation can be a mess.

2 A fix for this shit

As I argued multiple times, a fix for point 1 is adopting other voting procedures, such as range voting. We will then try to understand how to implement range voting with a tokens-and-shit approach.

I am giving for granted that you know how range voting works. If you don't, read about it on the other PDF or on Wikipedia. As an alternative, just fuck yourself.

Definition 2.1. Suppose we have a set $\mathscr{N}$ of candidates. Suppose moreover we have a set $\mathfrak{M}$ of possible voting values. For instance, if we have three people proposing (call them A,B,C) and the user can rate every one of these people from -3 to -3, then $\mathscr{N} := \{A,B,C\}$ and $\mathfrak{M} := \{-3,-2,-1,0,1,2,3\}$, hence $|\mathscr{N}| = 3$ and $|\mathfrak{M}| = 7$.

Submitted to:
Swarm group

We can express the voting of a user as a string (a,b,c), where $a,b,c \in \mathfrak{M}$. a represents the rating given to candidate A, b the rating given to candidate B and c the rating given to candidate C. This string is called *voting string*.

Definition 2.2. We denote an user A delegating his/her vote to a user B with $A \to B$.

2.1 The algorythm

Here we draft an algorythm to take care of voting. An analysis of this algorythm will be given in the next section.

2.1.1 Phase 1: Submitting candidatures

- A contract called **CANDIDATES** is created. This contract determines from when to when voting is open and is fundamentally a list of strings representing the candidates;
- An user that wants to run for the election writes a bio and a covering letter and he signs this with his/eth ethwallet private key.
- The user adds to a public contract a line like this:

$$(\text{ethwallet} \mid \text{hash of the bio} \mid \text{hash of the covering letter})$$

- When the window closes, every new submission is rejected.

2.1.2 Guardians and private keys

- A public key is created out of two secret keys, that will be kept by two different trusted parties (us and the tokensale guys, for instance);
- The public key is denoted with *pub*, while the secret keys are denoted with sec_1 and sec_2, respectively. It makes sense to give one of these secret keys to the Estonian non-profit organization that oversees Swarm, if possible.

2.1.3 Voting window

- Every person owning an eth wallet (even with no swarm tokens in it) can submit a vote. Every user can see who are the candidates running on our dedicated website. The website just displays the addresses of all the candidates along with their bios and covering letters, ordered by their stake. The fact that **CANDIDATES** contains this information ensures that we are not providing incorrect/manipulated information about the candidates.
- An eth contract, called **VOTE**, is created on the blockchain. This contract determines from when to when voting is open and is fundamentally a list of strings representing the users votes.
- The user vote is a string defined as follows:

$$(\text{user wallet address}, \text{voting string}, \text{delegation address})$$

Where:

user wallet address is the eth address of the voter;

voting string is a string defined as in Definition 2.1, where $\mathfrak{M}$ is the set of all the addresses in **CANDIDATES**. To be specific, the first entry of the voting string refers to the first candidate in **CANDIDATES**, the second to the second candidate in **CANDIDATES** and so on.

delegation address is an eth address.

vote strings and delegation addresses are set to some standard value (0 for eth address) and $(0,0,\ldots,0)$ for the vote string) if the user does not specify any value.

- The user encrypts this string using *pub* and adds the line to **VOTE**.

2.1.4 Outcome

- When the voting window closes, the outcome of the vote is calculated. The secret keys sec_1 and sec_2 are made public and added to **VOTE**. With this information, any user can decrypt the content of **VOTE**.
- The outcome is calculated as follows: First, we decrypt **VOTE** and we copy its content to **VOTE'**. **VOTE'** is basically a huge database with all the addresses of people who voted, their voting preference and the delegation address specified.
- We check that no one used the Swarm foundation address to vote. Since the foundation owns like 33% of the total swarm tokens, this would be very unfair: Partners can of course vote with their personal wallets but not with the foundation one. Hardcoding this guarantees transparency both for the partners and for the investors.

  We scroll **VOTE'** and check the first address of each string. If a string starting with the Swarm foundation eth address is found, we erase it. We keep doing this until we reach the bottom of the contract.
- After this, we have to check that people did not vote two times. This is easy: We scroll the **VOTE'** database again and check the first address of each string (voter's address). If two or more strings have the same voting address, only the most recent is kept. Since strings are added as votes come, we just have to erase all but the bottommost one.
- Then we check that all the delegations are correct. This means we have to rule out cases in which there are delegation loops, i.e. we have to deal with situations like $A \to B \to C \to A$ where we have a loop and it is not clear who is voting on behalf of who. To do this we apply the following algorithm:
  - Call $a,b,\ldots$ the entries in **VOTE'**. Starting from a, we form a string called *check* as follows:
    - The first entry of *check* is $a[1]$ (user wallet address).
    - The next entry of *check* is $a[3]$ (the delegation address).
    - If $a[3] = 0$ then the algorithm terminates and we run it on the next entry in **VOTE'**. Otherwise we scroll down **VOTE'** until we find an entry n such that $n[1] = a[3]$ (we check if the delegated address has voted). If this string does not exists, we then modify **VOTE'** setting $a[3] = 0$ (delegation is not valid). If it exists, we add $n[1]$ to *check*.
    - We re-run the same procedure starting from $n[1]$.
    - At every step, we check if *check* has repeated entries. Two things can happen:
      - If we find that repeated entries, then *check* will look like $(a,b,x,c,d,\ldots,f,x)$. In this case we modify **VOTE'** setting $x[3] = c[3] = \cdots = f[3] = 0$ (we eliminate the loop setting all the delegations in the loop as not valid); then we start again using the string immediately following a.

· If we don't find repeated entries, we eventually reach the end of **VOTE'**. In this case we start again using the string immediately following a.

– At the end of this process, **VOTE'** will be free of delegation loops. At this point we calculate the outcome as follows:
  * For every string n we check that $n[2]$ is a valid voting string, meaning that all the points assigned are in the correct range and the number of entries in the string equals the number of candidates. Mathematically, this means that every entry in $n[2]$ is in $\mathfrak{M}$ and that the lenght of n is $|\mathscr{N}|$. If this is not the case we set $n[2]$ to the string made of $|\mathscr{N}|$ entries in which every entry is 0. We will denote this string just with $\mathbf{0}$.
  * For every string n, call $\sigma_{n[1]}$ the stake of $n[1]$. If $n[3] = 0$ (no delegation), then $n[2] = \sigma_{n[1]} n[2]$, meaning that we multiply every entry of $n[2]$ by $\sigma_{n[1]}$. If $n[3] = a$, then $n[2] = \mathbf{0}$ and $\sigma_{k[1]} = \sigma_{n[1]} + \sigma_{k[1]}$ (we transfer the stake to the delegated user), where k is the entry such that $k[1] = a$ (having eliminated loops there is only one such string k).
  * We calculate the string *outcome* as $\sum_{n \in \mathbf{VOTE}} n[2]$ (we are summing all the points in all the voting strings componentwise).
  * The elected candidates are the ones with the highest amount of points in *outcome*. For instance, if $outcome = (234, 152, 36)$ then the candidate corresponding to the first string entry is the winner.
  * We have to elect three candidates for the board. We proceed as follows: We consider the three candidates having the biggest amount of points. If these are more than three, we further order them considering their stake, meaning that if A, B have the same amount of points but the stake of A is bigger than the stake of B, then A is preferred. If we still have more than three candidates ex-aequo, then we randomly chose three.

2.2 Analysis

With this algorithm we solve a shitload or problems. No one is able to see who voted for who and what is the delegation structure until the vote ends. The fact that the public key is obtained from two different secret keys means that not even the guardians are able to do this alone. The guardians should agree to fuck the platform sharing their secret keys and accessing the votes outcome while the voting window is still open. This is obviously possible and this is why the guardians should be two trusted and independent entities.

Users are able to delegate their vote to someone. If this delegation doesn't work well (loops, the delegate didn't vote) then the system falls back on the user preference. This means that if your delegate is a dickhead your vote is not wasted.

Since the stake is taken to be the one at the closing of the voting window, people cannot really fiddle with their token in the hope to increase their voting stake. Moreover, the voting is not made sending tokens to this or that address, and no tokens are locked in the process, so everyone can move his/her tokens as he/she pleases during a voting. The only thing to keep in mind is that your stake will be the one at the end of the closing window.