saphari

A. Abstract

Bitcoin relies on a double decentralization :

- There are non colluding full nodes all over the world which check integrity of the Txs and blocks : integrity of the ledger.
- There are non colluding miners which struggle against entropy : order of the ledger.

It brought several innovations :

- Foreseeable and limited money supply ; which is ideal to transfer value over time.
- Uncensurable, worldwide scalable, quite fast use of money ; which is ideal to transfer value over space.

But :

- Is this **guaranteed** ‘quasi-instantaneity’ of the Txs validation a good thing ? We question here the systematic discrepancy between the speed of the financial trades and the speed of the commercial trades.
- Would it be a good thing to have only one universal money ? We question here the use of a largely common token, considering the subjective theory of value as true.

We will not discuss these questions in this paper. But, this paper decribes an algorithm which could suit to people answering ‘no’ to these questions.

We imagined another distributed consensus algorithm where :

- Users are collectively deciding the speed of Txs validation.
- Launching a new cryptocurrency requires a minimal ecosystem : users owning computers reliably connected together.
- Doing Txs is free of commission.

The proposed system is UTXO based and relies on a DAG where each node represents a Tx. Nodes are linked each others as Txs must point to (or ‘approve’) previous Txs to be attached to the DAG. Txs get validated according to the longest branch rule.

The limitations of this algorithm are :

- Attacking the algorithm would require to buy 50% of the whole monetary supply.
- The algorithm finds a consensus as long as 50% of the whole monetary supply is still spent by honnest users.

Unlike POS algorithm, users participating to the consensus are not getting wealthier.

B. Consensus rules

1. A Tx T offers a number of empty slots equal to its amount and must point to a number of empty slots (of already existing Txs) equal to its amount . By pointing to previous Txs, the Tx T is approving them.

2. To measure the approval weight of a Tx T, the algorithm counts all the Txs of a branch (including double, triple…spent Txs) which are pointing directly or indirectly to T. This calculation doesn't matter the number of slots pointed : all or none principle.

> For example if Ta (1000 tokens) points to only one slot of Tb (1000 tokens also) which points to 100 slots of Tc (1000 tokens), the approval weight of Tc is 2000 (1000+1000).

3. A Tx is spendable once its approval weight is bigger than 0.5*M where M is the money supply.

> The users must so continuously do Txs (to themselves if they want to 'hodl') to validate past Txs. As a consequence, the users commonly control the current validation time : from ∞ to the network limit.

4. A spendable Tx T can be spent only if it points (directly or indirectly) to at least one slot of a Tx which is giving to T an approval weight bigger than 0.5*M.

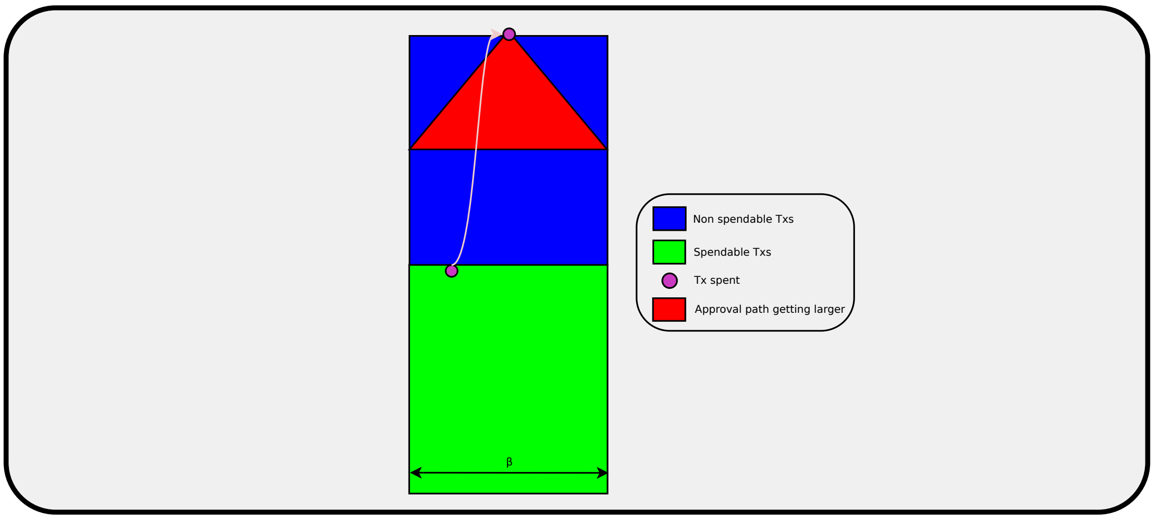
5. The genesis Tx contains all the tokens at start-up : M. Genesis Tx is 'at birth' validated and offers a limited number of empty slots 'β'. β is the width of the DAG. β is also the maximum amount of any Tx.

> We so have a DAG, with a constant width (β slots) over time, which is getting higher and higher thanks to fresh Txs pointing to the empty slots of previous Txs.

6. There is a minimal amount of the Txs. A Tx below this amount will not be relayed by the servers. We will call this minimal amount 'α'.

> The amount of a Tx must so be between α (minimum value) and β (maximum value)

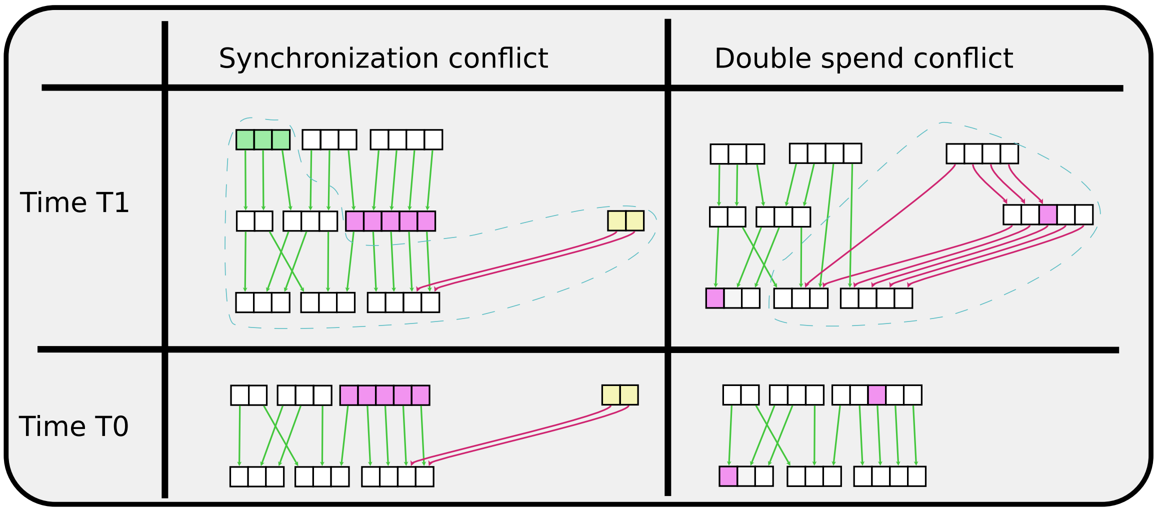
7. A Tx points to other Txs (stage n=1), which points to other Txs (stage n=2), which points to other Txs (stage n=3),etc…between 2 stages the approval weight must be multiplied by α or more. The goal of this rule is to force any new Tx to approve the whole width of the DAG with as few stages as possible. Why α ? Because, considering a DAG composed only of the smallest Txs possible, at each stage, we can't make the approval weight growing faster.



> The purple Tx became spendable thanks to the last Tx on the top of the red triangle (not represented). According to the rule 4, the purple Tx can be spent if it points to this approving Tx or 'above'. Thanks to this rule, all users can well check that this Tx was really spendable when it was spent.

8. It can happen (very often) that 2 conflicting branches of the DAG appear. The reasons can be :

- One (or more) token has been double spent in two different Txs
- Two Txs are pointing directly to at least one common slot



> Synchronization conflict : The yellow and the red Txs are directly pointing to the two same slots. These two Txs are in conflict. At time T1, some users added 3 Txs. Only the green Tx has been attached to the DAG avoiding choosing if the red or the yellow Tx is legit. Because of rule 7, it will soon not be possible to avoid choosing : Two fully distinct branches are appearing.

> Double spend conflict : The red token has been spent two times. Two distinct branches are appearing.

9. Each time two branches are appearing, honnest users should consider as legit the branch they saw first. They should never switch, for a first spend, to the branch with the

heaviest weight. If, too much often, the first branch seen is not the branch adopted by the network, the user's server should establish connection with new peers.

> Without this rule, with a small amount of money, it could be possible for a user A to 'censor' another token : when a Tx T, spending this token, is attached to the DAG, the user A would just have to point to some common slots with the Tx T. He so creates a synchronization conflict. He then just has to give more weight to the branch he is deliberatly creating and all other users would follow the censoring branch.

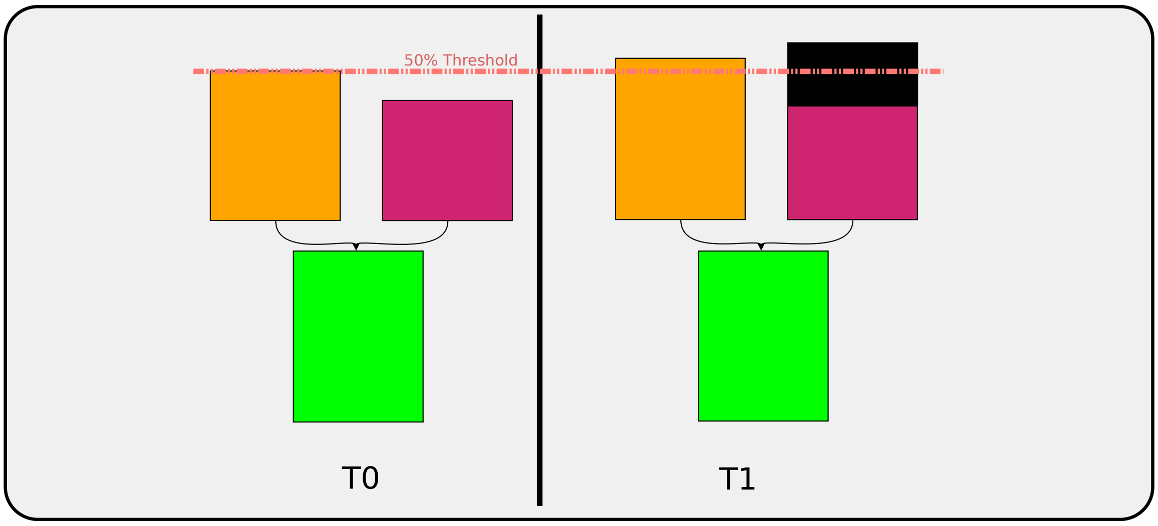
10. The receiver decides the empty slots of the existing Txs which will be directly pointed. It's his own interest to well integrate the Txs to the DAG to well get the tokens validated.

> Practically, it means that, to do a Tx, the receiver shows to the sender enough data to cryptographically identify its own public key and the slots to point to.

11. Relaying rules limit Tx spamming and, when possible, take advantage of the absolute ordering offered by the DAG. Each time a server receives a Tx :

- It checks its integrity. If the Tx is not OK, it will not be relayed.
- It checks that the Txs pointed are well in the DAG, otherwise this Tx will not be relayed and stays in a local mempool till the server integrates in the DAG all the Txs pointed.
- A Tx which is pointing directly or indirectly to two conflicting branches will not be realyed.
- A Tx double spending a token can not point directly or indirectly to the first Tx spending this token.

C. Delaying the consensus



At time T0, the orange branch has reached enough weight : The rule 3 is fullfilled and one of the 2 conflicting Txs can be spent. This Tx will be spent at the top of the orange branch (according to rule 4). The honnest users are so spending in the orange branch.

During time T0, the attacker is increasing the weight of the red branch to give "spending right" to the other conflicting Tx **and** to overcome the orange branch. If he still has non spent tokens, he will spend them in the red branch. Otherwise, he has to double spend tokens already spent in the orange branch.

At time T1, the red branch is the branch which has the the biggest approval weight towards the 2 conflicting Txs. Honnest users stop spending in the orange branch and spend so in the red branch.

The attacker has no choice to continue the attack : he has to have simple spent tokens in the red branch and double spend them in the orange branch. Honnest users will have to stop spending in the orange branch and to spend again in the red branch.

We can repeat this story till the attacker gets run out of tokens.

The only possibility for the attacker to delay indefinitely the consensus, is to have 50% of all tokens. If he has 50% of all tokens, then he can make a branch progressing indefinitely, doesn't matter what the honnest users do.

Considering all users have, at each moment, the same view of the DAG (which is wrong assumption due to network latency), once one of the two conflicting Txs has been spent two times, we can consider it as validated.

Actually, considering network latency, a Tx can be considered as validated as soon as this Tx belongs to the longest branch and that this branch is overcoming all other branches, by at least 50% of M.

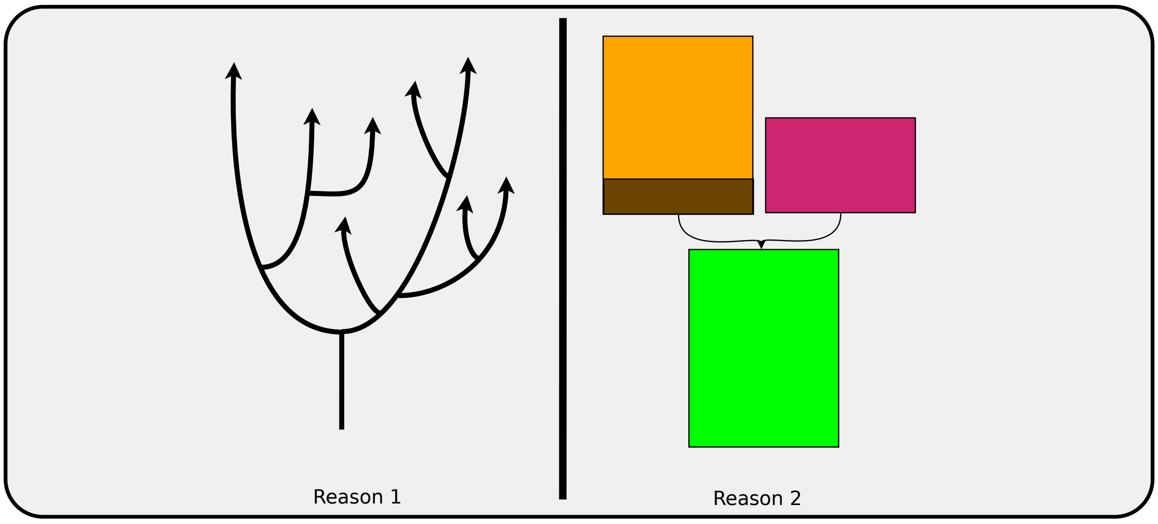
Up to now, in the scenario described above, honnest users have always simple spent tokens, following the 'influence' of the attacker on the DAG. When the attack has passed (one branch is overcoming all others by more than 50% of M), honnest users with tokens spent in the non-legit branch will double spend these tokens into the 'winning' branch : they want to get back into the race and continue participating in the consensus. These double spent Txs are not malicious.

The good behaviour of honnest users (not feeding the 2 branches at the same time) helps to get a consensus faster. But as long as their behaviour is honnest, even if some honnest users double spend some tokens, a consensus on the longest branch will be found. The attacker can only delay the consensus.

D. When honnest users double spend

After a defined period P, without any new Tx attached to the DAG, honnest users will double spend tokens.

Two reasons (that can be combined), can explain why the DAG could get frozen.



> Reason 1 : Either because of too many conflicts; no one branch reached the critical approval weight : 50% of M.

> Reason 2 : Because some users are not spending their tokens. In the picture above, the orange branch has reached the critical approval weight (50% of M) but the greyed area represents tokens hold by an attacker. He is deliberatly not spending his tokens.

After the period P, honnest users should double spend first the tokens located in the branch with the lowest approval weight (towards the oldest non solved conflict). Then they wait again the period T, and repeat the operation till the DAG can again progress thanks to all users.

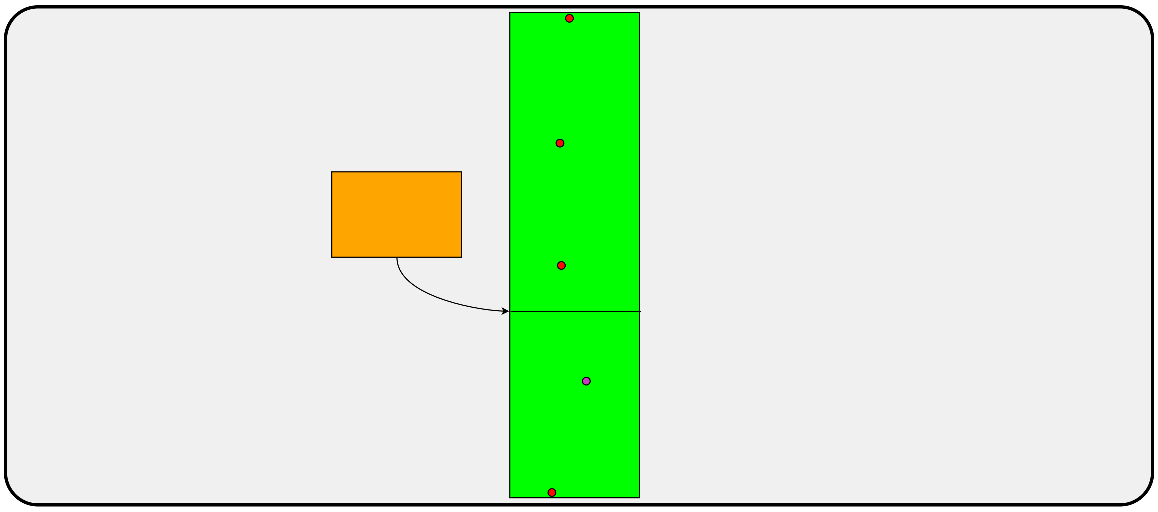
If the DAG gets definitely frozen (50% of the tokens are no more spent), a new DAG should be launched with all tokens which don't have the right to be spent (preserving the keys of these users). Unfortunately, it requires a central authority.

E. Old spenders attack

A potential attacker could ask for private keys of former users of this cryptocurrency. As former users are not anymore "in the game", they could accept to communicate their private keys : They don't care anymore about the reliability of the algorithm.

Actually, to do so, the attacker has to convince, the token holders who represent 50% of M, **at a defined period**. Why that ?

Because of these 2 rules :
- The rule 4, says that to be spent a Tx T must point (directly or indirectly) to at least one Tx giving to T an approval weight above 50% of M. - The rule 11, says that a Tx double spending a token can not point directly or indirectly to the 'first spend' Tx spending this token.



> The circles represent the Txs of a user who is spending his tokens to himself, as soon as he can spend again. Only the purple Tx could be attached to the attacking branch.

F. Conclusion

We proposed a commision free and fully distributed consensus algorithm applicable to any token with value for humans. The network is robust in its unstructured simplicity :D

It’s not worldwide scalable as it’s likely not possible to build a second layer on it. But the idea would be more a multitude of cryptocurrencies, exchangeable thanks to atomic swaps, where, to get protected against speculation attacks, users could play with validation time.

G. License and author

The SHA256 of the PDF of this repository will be time-stamped in Bitcoin blockchain.