Local modifiers in the Tangle

Serguei Popov[1,*]

March 17, 2018

[1]Department of Statistics, Institute of Mathematics, Statistics and Scientific Computation, University of Campinas – UNICAMP, rua Sérgio Buarque de Holanda 651, 13083–859, Campinas SP, Brazil
e-mails: popov@ime.unicamp.br and serguei.popov@iota.org

Abstract

As of today, there are many cryptocurrency systems around. They mostly share one common feature: all participants of the network interpret the same ledger in the same way. In this paper, we introduce the idea of *local modifiers*: the nodes of the network can interact with the ledger in different ways, depending on various kinds of information locally available to them. We then argue that such an approach permits to increase the security and the scalability of the Tangle.

1 Introduction

In general, cryptocurrency systems are assumed to work in the following way. At each time $t > 0$, the network is composed by finitely many *nodes*. For definiteness, we assume that all possible nodes are indexed by natural numbers[1], and $\mathsf{St}(n) \in \mathfrak{S}$ stands for the *state* of the node n (i.e., if it exists or not, online/offline etc.); for now, we assume that all possible nodes' states are elements os a finite set $\mathfrak{S}$. We also denote by $\Lambda_t \subset \mathbb{N}$ the set of nodes that are online at time t, i.e., $\Lambda_t = \{n \in \mathbb{N} : \mathsf{St}(n) = \text{“online”}\}$. Next, at each time $t > 0$ there is a data set $\mathsf{Ledger}(t)$ describing the whole history of the system up to that time moment. *In principle*, it is stored in

*corresponding author

[1]in this paper, we use the notation $\mathbb{N} = \{1, 2, 3, \ldots\}$ for the natural numbers, and $\mathbb{Z}_+ = \{0, 1, 2, \ldots\}$ for positive integers

all nodes of the network, and is the same for everyone. This ledger is incremental, in the sense that $\mathsf{Ledger}(t) \subset \mathsf{Ledger}(s)$ for all $t < s$; that is, data is added but never deleted.

Now, let us make a simplifying assumption that the ledger is a (growing) set of *transactions*[2]. Basically, a node must be able to do three things:

(a) Determine if a given subset of the ledger is *valid* (no conflicting transactions etc., let's not elaborate on it, for now).

(b) Determine the *confidence level* $\gamma_t(x) \in [0, 1]$ of a given transaction $x \in \mathsf{Ledger}(t)$.

(c) Know what to do in order to issue a transaction. (in Bitcoin it's clear, but in IOTA it must do PoW and know where to attach).

Let us elaborate on item (b) above. We do not yet specify what exactly this confidence level is, apart from mentioning that $\gamma_t(x) = 1$ means that transaction x must be accepted (i.e., considered to be confirmed), while $\gamma_t(x) = 0$ means that transaction x must not be accepted. As an example, the usual rule "accept after six confirmations" in Bitcoin can be translated to this notation as $\gamma_t(x) = 1$ if transaction x has at least six confirmations at time t, and $\gamma_t(x) = 0$ otherwise. Of course, intermediate (between 0 and 1) confidence levels are also possible. (However, we have to observe that the confidence level need not be necessarily equal to the probability that the transaction will eventually be confirmed; rather, it is just the result of application of some pre-defined rule.) What is important to observe, is that $\gamma_t(x)$ is *completely determined* by the state of the ledger at time t: there exists a (deterministic) function Γ such that

$$\gamma_t(x) = \Gamma(x, \mathsf{Ledger}(t)). \tag{1}$$

As for the item (c), the situation is similar: if a node $m \in \Lambda_t$ has to issue a transaction at time t, the action $\mathsf{Act}^{(m)}(t)$ of the node at the moment t is determined by the current state of the ledger together with the properties of account m (such as private keys etc.): for a deterministic function Ψ,

$$\mathsf{Act}^{(m)}(t) = \Psi(m, \mathsf{Ledger}(t)). \tag{2}$$

In the case of IOTA, this means that a node chooses the probability distribution of the pair of attachments' locations based only on the current state of the ledger

[2]in general, there may also be smart contracts or any other sorts of data, but we focus on transactions to make the exposition cleaner

(here we are discussing only the "honest" nodes, i.e., the nodes that use the default attachment strategy; see [3] for definitions and explanations).

Let us think about the drawbacks of the above approach. For example:

- The assumption that all online nodes have $\mathsf{Ledger}(t)$ at time t is clearly impractical because of the propagation delays. For the same reason, it can hardly be true that all nodes have the same ledger at any given time[3].
- Also, the requirement that all nodes store the whole history may be problematic, simply because of limited hard disk space.
- Another important observation is that any person or entity willing to attack the network, can actually *predict* the behavior of any node. That is, the attacker knows which actions other nodes will take if he e.g. broadcasts a given set of (possibly malicious) transactions, and which transactions they will consider as confirmed in such circumstances. Example: parasite chain attack, cf. Section 4.1 of [2].

1.1 Local modifiers

Now, we describe the main idea of this paper. Let us examine the above-mentioned parasite chain attack (see Figure 1). The attacker prepares a secret subtangle containing a transaction x which spends all funds from his address a_1 to another address a_2 also controlled by him. In the meanwhile, from the same address a_1, he pays to a merchant (transaction y on Figure 1), transferring some amount to an address b_1 controlled by the merchant. The attacker then waits until the merchant delivers the goods, and after that broadcasts his secret subtangle, hoping that its weight is enough to force the tip-selecting walks of the honest nodes typically end in one of the attacker's tips. This will make the attacker's subtangle outgrow the legitimate one, and therefore the double-spending transaction will be confirmed, while the legitimate one will be dropped by the network.

The key observation is that the attacker must first *wait*, and only after that reveal his secret subtangle all at once. That is, this secret subtangle will arrive *later than it normally would* to honest nodes. Now, assume that the honest nodes use the

[3]it may be approximately true for cryptocurrencies with infrequent blocks, such as the Bitcoin (and only at times when the last block has sufficiently propagated), but we cannot have anything like this in any globally scalable distributed cryptosystem

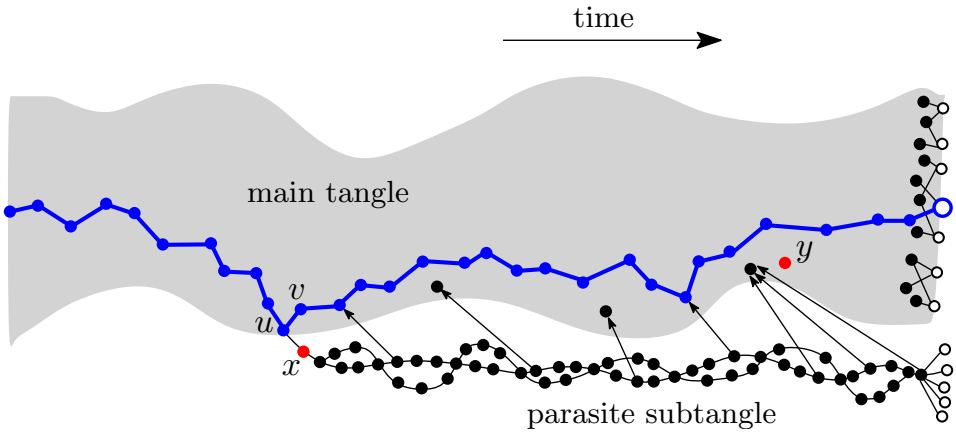

Figure 1: An illustration of an attempted double-spend. The transaction x spends the funds that the attacker wants to spend once more in transaction y, before revealing the subtangle containing x. The thick blue line corresponds to a trajectory of a typical tip-selecting random walk (with $q = 0$; see (7) below). (The grey area contains many transactions which are not shown in order to keep the picture clean.)

following rule: if a transaction arrives later than it should[4], then "penalize" it by e.g. reducing its (own) weight, or in some other way (we'll discuss it below).

What would happen then? Well, the honest nodes have made *local modifications* by decreasing the weights of attacker's transactions, thus causing their tip-selecting random walk to avoid the attacker's subtangle, causing the double-spending attack to fail.

Another important observation is that the attacker typically cannot know the local modifiers of the honest nodes (at least if the honest nodes exchange the information via encrypted channels), and so the attacker cannot predict the honest nodes' behavior. A consequence of this is that it becomes more difficult to attack the network.

More formally, for $m \in \Lambda_t$ let us denote by $\mathsf{Mdf}^{(m)}(t)$ the set of local modifiers of the node m at time t; we put $\mathsf{Mdf}^{(m)}(t) = \emptyset$ in the case when there are no modifiers at all. We then denote by $\mathsf{Ledger}(t) \oplus \mathsf{Mdf}^{(m)}(t)$ the *locally modified* (by node m) ledger at time t, with the convention $\mathsf{Ledger}(t) \oplus \emptyset = \mathsf{Ledger}(t)$. The new versions of (1)–(2) are then

$$\gamma_t^{(m)}(x) = \Gamma\big(x, \mathsf{Ledger}(t) \oplus \mathsf{Mdf}^{(m)}(t)\big), \tag{3}$$

[4]in Section 2 we elaborate on the meaning of this

and

$$\mathsf{Act}^{(m)}(t) = \Psi\big(m, \mathsf{Ledger}(t) \oplus \mathsf{Mdf}^{(m)}(t)\big). \tag{4}$$

For formal reasons, we also consider a "special" (virtual) node with index 0. This node has no local modifiers, i.e., $\mathsf{Mdf}^{(0)}(t) = \emptyset$ for all $t \geq 0$. Let us also define $\Lambda_t^* = \Lambda_t \cup \{0\}$.

Now, the problem is that, despite the fact that the nodes view the ledger differently, they still need to achieve consensus on the validity of transactions. What we need, is that, for all $t_0 \geq 0$,

$$\mathbb{P}\Big[\lim_{t\to\infty} \gamma_t^{(m)}(x) \text{ exists and is the same for all } m \in \mathbb{Z}_+\Big] = 1, \quad \text{for all } x \in \mathsf{Ledger}(t_0). \tag{5}$$

Note that[5] (5) means that the newcoming nodes, which have no local modifiers at the moment they join the system, are nevertheless able to access correctly the validity of transactions (except maybe for the most recent ones).

Besides proving (5), it would be useful to obtain some quantitative statements, e.g. on the speed of convergence. Generally, purely qualitative statements in the crypto context may be misleading: e.g., one can "prove" that the Bitcoin (as well as anything else) is broken simply by arguing that the probability of guessing the private key of an address is strictly positive, and so, *eventually*, someone will guess it a.s.

Our goal is then to introduce some concrete rules of doing local modifications, in such a way that (5) holds.

1.2 More definitions and notations

We now recall some definitions and notations of [3]. Define $\mathrm{card}(A)$ to be the cardinality of (multi)set A. For an oriented graph $\mathcal{T} = (V, E)$, where V is the set of vertices and E is the multiset of edges, and $v \in V$, let us denote by

$$\deg_{\mathrm{in}}(v) = \mathrm{card}\{e = (u_1, u_2) \in E : u_2 = v\},$$
$$\deg_{\mathrm{out}}(v) = \mathrm{card}\{e = (u_1, u_2) \in E : u_1 = v\}$$

the "incoming" and "outgoing" degrees of the vertex v (counting the multiple edges). In what follows, we refer to multigraphs simply as graphs. The vertices of the tangle are also called transactions. For $u, v \in V$, we say that u *approves* v, if $(u, v) \in E$. We use the notation $\mathcal{A}(u)$ for the set of the vertices approved by u. We say that $u \in V$

[5] recall that $0 \in \mathbb{Z}_+$

references $v \in V$ if there is a sequence of sites $u = x_0, x_1, \ldots, x_k = v$ such that $x_j \in \mathcal{A}(x_{j-1})$ for all $j = 1, \ldots, k$ (i.e., there is a directed path from u to v). If $\deg_{\text{in}}(w) = 0$ (i.e., there are no edges pointing to w), then we call the vertex $w \in V$ a *tip*.

Let $\mathcal{G}$ be the set of all Directed Acyclic Graphs (also known as DAGs, that is, oriented graphs without cycles) $G = (V, E)$ with the following properties:

- the graph G is finite and the multiplicity of any edge is at most two;
- there is a (unique) distinguished vertex $\wp \in V$ such that $\deg_{\text{out}}(v) = 2$ for all $v \in V \setminus \{\wp\}$, and $\deg_{\text{out}}(\wp) = 0$ (this vertex $\wp$ is called *the genesis*);
- any $v \in V$ such that $v \neq \wp$ references $\wp$; that is, there is an oriented path[6] from v to $\wp$ (one can say that the graph is *connected towards* $\wp$).

The tangle is a continuous-time Markov process on the space $\mathcal{G}$. The state of the tangle at time $t \geq 0$ is a DAG $\mathcal{T}(t) = (V_{\mathcal{T}}(t), E_{\mathcal{T}}(t))$, where $V_{\mathcal{T}}(t)$ is the set of vertices and $E_{\mathcal{T}}(t)$ is the multiset of directed edges at time t. The process's dynamics are described in the following way:

- The initial state of the process is defined by $V_{\mathcal{T}}(0) = \wp$, $E_{\mathcal{T}}(0) = \emptyset$.
- The tangle *grows with time*, that is, $V_{\mathcal{T}}(t_1) \subset V_{\mathcal{T}}(t_2)$ and $E_{\mathcal{T}}(t_1) \subset E_{\mathcal{T}}(t_2)$ whenever $0 \leq t_1 < t_2$.
- For a fixed parameter $\lambda > 0$, there is a Poisson process of incoming transactions; these transactions then become the vertices of the tangle.
- Each incoming transaction chooses[7] two vertices v' and v'' (it is possible that $v' = v''$), and two oriented edges from v to v' and v'' are added[8].
- Specifically, if a new transaction v arrived at time t', then $V_{\mathcal{T}}(t'+) = V_{\mathcal{T}}(t') \cup \{v\}$, and $E_{\mathcal{T}}(t'+) = E_{\mathcal{T}}(t) \cup \{(v, v'), (v, v'')\}$.

Let us write

$$\mathcal{P}^{(t)}(x) = \big\{y \in \mathcal{T}(t) : y \text{ is referenced by } x\big\},$$
$$\mathcal{F}^{(t)}(x) = \big\{z \in \mathcal{T}(t) : z \text{ references } x\big\}$$

[6] not necessarily unique

[7] the precise selection mechanism will be described below

[8] we say in this case that this new transaction was *attached* to v' and v'', or, equivalently, v *approves* v' and v''

for the "past" and the "future" with respect to x (at time t). Observe that these introduce a *partial order* structure on the tangle. Notice that, if t_0 is the time moment when x was attached to the tangle, then $\mathcal{P}^{(t)}(x) = \mathcal{P}^{(t_0)}(x)$ for all $t \geq t_0$. We also define the *cumulative weight* $\mathcal{H}_x^{(t)}$ of the vertex x at time t by

$$\mathcal{H}_x^{(t)} = 1 + \operatorname{card}\bigl(\mathcal{F}^{(t)}(x)\bigr) = 1 + \sum_{y \in \mathcal{F}^{(t)}(x)} 1; \tag{6}$$

that is, the cumulative weight of x is one (its "own weight") plus the number of vertices that reference it.

There is some data associated to each vertex (transaction), created at the moment when that transaction was attached to the tangle; we assume that it is an element of some (unspecified, but finite) set $\mathcal{D}$. It is important to note that there is a natural way to say if the set of vertices is *consistent* with respect to the data they contain. To emphasize that the vertices of $G \in \mathcal{G}$ contain some data, we consider the *marked* DAG $G^{[\mathfrak{d}]}$ to be $(G, \mathfrak{d}) = (V, E, \mathfrak{d})$, where $\mathfrak{d}$ is a function $V \to \mathcal{D}$. Let us define $\mathcal{G}^{[\mathfrak{d}]}$ to be the set of all marked DAGs $(G, \mathfrak{d})$, where $G \in \mathcal{G}$.

It is important to observe that the way how a node chooses two transactions to approve is not *enforced*. However, the *recommended* algorithm for this may be described in the following way. Assume that a node is about to issue a transaction, and that the state of the tangle that it observes is $(G, \mathfrak{d})$ (note that G need not be equal to the *real* state of the tangle at that time, since the node may not be fully synchronized with the network). Let us denote by $\mathcal{L}$ the set of all vertices that are tips in G. We then define the *tip-selecting random walk*, in the following way. It depends on a nonnegative parameter q (the backtracking probability) and on a function f. The initial state of the random walk is the genesis $\wp$, and it is stopped upon hitting the set $\mathcal{L}$. Let $f : \mathbb{R}_+ \to \mathbb{R}_+$ be a monotone nonincreasing function. The transition probabilities are defined in the following way: the walk *backtracks* (i.e., jumps to a randomly chosen site it approves) with probability $q \in [0, 1/2)$; if y approves $x \neq \wp$ ($y \rightsquigarrow x$), then the transition probability $P_{xy}^{(f)}$ is proportional to $f(\mathcal{H}_x - \mathcal{H}_y)$, that is

$$P_{xy}^{(f)} = \begin{cases} \dfrac{q}{2}, & \text{if } y \in \mathcal{A}(x), \\ \dfrac{(1-q) f(\mathcal{H}_x - \mathcal{H}_y)}{\sum_{z : x \in \mathcal{A}(z)} f(\mathcal{H}_x - \mathcal{H}_z)}, & \text{if } x \in \mathcal{A}(y), \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

(for $x = \wp$ we define the transition probabilities as above, but with $q = 0$).

Let us comment on (7), to get some intuition on what the above random walk really does. In average (since $q < 1/2$) the random walk "prefers" to go towards the tips. Now, when it makes a jump toward the tips (cf. the second line in (7)), it "prefers" to go to the site which minimizes the difference of cumulative weights. For instance, let us go back to Figure 1 and assume that the walker is currently in u, and the parasite subtangle have been published already. Then, typically, the cumulative weight of x will be *much* less than that of v, and so, according to (7), the walk is much more likely to jump to v than to x. One can also observe that, in the case when f decays *very* rapidly and $q = 0$, the random walk converges to the path chosen by the GHOST protocol [4].

The selection of two tips $w_{1,2}$ where a newcoming transaction will be attached is done using the following two-step procedure. First, we use the above random walk with some (very) rapidly decaying function f_0 (for example, $f_0(s) = e^{-\alpha s}$ with a large $\alpha > 0$) in order to choose a "model tip" w_0. Then, using random walks with a "moderately" decaying function f_1 (for example, $f_0(s) = s^{-3}$) we find $w_{1,2}$ as the tips where they stopped, also checking that

$$\mathcal{P}(w_0) \cup \mathcal{P}(w_1) \cup \mathcal{P}(w_2)$$

is consistent (if not, then discard the newly chosen tips and rerun the random walk). One can also require that w_1 should be different from w_2; for that, one may re-run the second random walk in the case its exit point happened to be the same as that of the first random walk.

We say that a transaction is *confirmed with confidence* γ_0 (where γ_0 is some predefined number, close to 1), if, with probability at least γ_0, the random walk with (rapidly decaying f_0) ends in a tip which references that transaction.

2 Constructing the local modifiers

There are, of course, many possible ways to construct *reasonable* local modifiers. In the following, we describe one basic idea that, apparently, works well against both parasite chain attack and splitting attack. Let us denote by $t_v^{(m)}$ the time when the node m received the transaction v. Notice that, for a given m, the dataset $(t_v^{(m)})$ need not be necessarily consistent: if u references v, $t_u^{(m)}$ need not be smaller than $t_v^{(m)}$.

Now, assume that a given node m was online during a relatively long time, and "believes" that it has a reasonably honest picture of the network. It may then consider choosing tips not with the random walk defined by (7), but with the following one. Let $g : \mathbb{R}_+ \to \mathbb{R}_+$ be a monotone nonincreasing function (it may, although does not

necessarily need to, be equal to the function f used in (7)). Analogously, let $q' \in [0, 1/2)$ be the backtracking probability. The (new) transition probabilities are defined in the following way:

$$\widehat{P}_{xy}^{(m,g)} = \begin{cases} \dfrac{q'}{2}, & \text{if } y \in \mathcal{A}(x), \\ \dfrac{(1-q')g\big((t_y^{(m)} - t_x^{(m)})^+\big)}{\sum_{z:x\in\mathcal{A}(z)} g\big((t_z^{(m)} - t_x^{(m)})^+\big)}, & \text{if } x \in \mathcal{A}(y), \\ 0, & \text{otherwise,} \end{cases} \tag{8}$$

where $(a)^+ := \max\{0, a\}$ is the *positive part* of $a \in \mathbb{R}$ (again, for $x = \wp$ we define the transition probabilities as above, but with $q' = 0$). The basic idea is that, if y approves x, but the former one was received *much* later than the latter, the random walker would typically avoid jumping from x to y. This provides some good defense against the parasite chain attack, because the attacker needs to keep the parasite chain hidden for some time, which means that the (local) time differences in the place where the parasite chain is attached to the main tangle would be rather large.

Another advantage of the abova approach is that the cumulative weight calculation may be computationally costly; a node using the random walk defined by (8) does not have to do this task.

Of course (and that may actually be a good idea), one can also use some "combination" of (7) and (8) to define the transition probabilities (i.e., use both cumulative weights and the local modifiers).

Then, let us describe how this approach defends against the splitting attack (same idea). The attacker wants to maintain the balance between two conflicting branches. Denote by $W_1(t), W_2(t)$ their weights at time $t \geq t_0$ (t_0 is the time when the attack starts), see Figure 2. Grosso modo, he needs to maintain $W_1 \approx W_2$. He checks the values at times $t_n = t_0 + \delta n$. However, he sees $W_1(t_n) + \xi_1^{(n)}$ and $W_2(t_n) + \xi_2^{(n)}$, where $(\xi_{1,2}^{(n)}, n \geq 1)$ are i.i.d. random variables (errors), let us assume them to be Normal(0,σ^2), with $\sigma \gg 1$.

The idea is that it is better for the attacker to send transactions in batches. So he needs to "store" transactions. But then the local modifiers will devalue the stored transactions.

Some good math will be inserted to this place :) In particular, we will use the Lyapunov functions method [1].

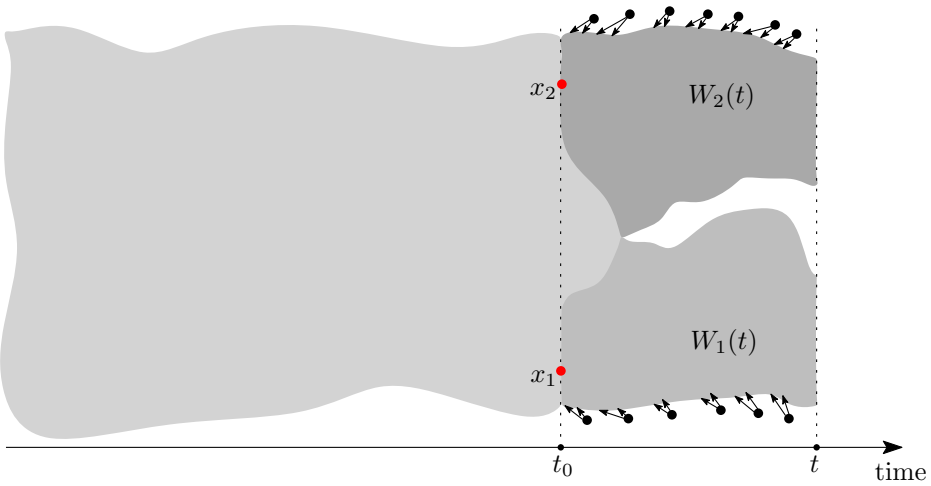

Figure 2: Splitting attack.

References

[1] M. Menshikov, S. Popov, A. Wade (2017) *Non-homogeneous Random Walks: Lyapunov Function Methods for Near-Critical Stochastic Systems.* Cambridge University Press, Cambridge.

[2] S. Popov (2015) The Tangle. `https://iota.org/IOTA_Whitepaper.pdf`

[3] S. Popov, O. Saa, P. Finardi (2017) Equilibria in the Tangle. arXiv:1712.05385

[4] Y. Sompolinsky, A. Zohar (2013) Secure high-rate transaction processing in Bitcoin. `https://eprint.iacr.org/2013/881.pdf`