The tangle: some aspects of a blockchainless cryptocurrency

Serguei Popov*

December 28, 2015

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-mail: `popov@ime.unicamp.br`

Abstract

In this paper we discuss some (mainly probabilistic) aspects of a cryptocurrency that, instead of a blockchain, uses a *tangle* (that is, a directed acyclic graph) for storing transactions. In particular, one of contributions of this paper is a family of MCMC algorithms for selecting the sites of the tangle where a newly arrived transaction should be attached.

1 Introduction

The rise and success of Bitcoin during the last six years proved the value of blockchain technology. However, this technology also has a number of drawbacks, which prevent it to be used as a one and only global platform for cryptocurrencies. Among these drawbacks, an especially notable one is the impossibility of making micro-payments, which have increased importance for the rapidly developing Internet-of-Things industry. This justifies a search for solutions essentially different from the blockchain technology, on which the Bitcoin and many other cryptocurrencies are based. In this paper we discuss a blockchainless approach, which is currently being implemented in the cryptocurrency system called *iota* [1], recently designed as a cryptocurrency for the Internet-of-Things industry. We, however, focus more on general features and

*a.k.a. `mthcl` at `bitcointalk.org` and `nxtforum.org` problems/difficulties that arise than one attempts to get rid of the blockchain, than on concrete details of iota's implementation.

In general, a tangle-based cryptocurrency works in the following way. As mentioned before, there is no global blockchain; on its place there is a DAG (= directed acyclic graph) that we call *tangle*. The transactions issued by nodes constitute the site set of the tangle. Its edge set is obtained in the following way: when a new transaction arrives, it must *approve* two[1] previous transactions; these approvals are represented by directed edges, as shown on Figure 1 and others (on the pictures, times always goes from left to right). If there is a directed path of length at least two from transaction A to transaction B, we say that A *indirectly approves* B. It is assumed that the nodes check if the approved transactions are not conflicting and do not approve (directly or indirectly) conflicting transactions. The idea is that, as a transaction gets more and more (direct or indirect) approvals, it becomes more accepted by the system; in other words, it will be more difficult (or even practically impossible) to make the system accept a double-spending transaction.

In the subsequent sections, after introducing some notations in Section 2, we discuss algorithms for choosing the two transactions to approve, the rules for measuring the overall transaction's approval (Section 3 and especially Section 3.1), and possible attack scenarios (Section 4).

Also, it should be noted that the ideas about usage of DAGs in the cryptocurrency context were around for some time, see e.g. [2, 3, 4, 5]. Especially, observe that in the work [3] a solution similar to ours was proposed.

This paper is a revised and shortened version of [6].

2 Weights and more

Here, we discuss the (own) weight of a transaction and related concepts. The weight of a transaction is proportional to the amount of hashing work that the issuing node invested into it; one can also consider situations when the weight is obtained in a PoS-like way, or even set to constant. Mostly, here for us it is irrelevant how the weight was obtained in practice; it is only important that every transaction has a positive number (= weight) attached to it. In general, the idea is that the larger the weight is, the more "important" the transaction is for the tangle. Usually it is assumed that, to avoid spamming and different kinds of attacks, no entity can generate too many

[1] this is the simplest approach; one may also study similar systems where transactions must approve k other transactions for a general $k \geq 2$, or consider more complicated rules

transactions with “acceptable” weights in a short time period. In fact, to understand the arguments in this paper, one may safely assume that all weights are equal to 1.

One of the notions we need is the *cumulative weight* of a transaction: it is defined as the own weight of this transaction plus the sum of own weights of all transactions that approve our transaction directly or indirectly. This algorithm of cumulative weights calculation is illustrated on Figure 1. The boxes represent transactions; the small numbers in the SE corner stand for the own weights of the transactions, while the (bigger) bold numbers are the cumulative weights. For example, the transaction F is approved, directly or indirectly, by the transactions A, B, C, E. The cumulative weight of F is $9 = 3 + 1 + 3 + 1 + 1$, that is, the sum of the weight of F and the weights of A, B, C, E.

On the top picture, the only unapproved transactions (the “tips”) are A and C. When the new transaction X comes and approves A and C, it becomes the only tip; the cumulative weight of all other transactions increases by 3 (which is the weight of X).

For the discussion of approval algorithms, we need also to introduce some other variables. First, for a site (i.e., a transaction) of the tangle, we introduce its

- *height*, as the length of the longest oriented path to the genesis;
- *depth*, as the length of the longest reverse-oriented path to some tip.

For example, on Figure 2, G has height 1 and depth 3 (because of the reverse path F, B, A), while D has height 2 and depth 2. Also, let us introduce the notion of the *score*. By definition, the score of a transaction is sum of own weights of all transactions approved by this transaction plus the own weight of the transaction See Figure 2. Again, the only tips are A and C. Transaction A approves (directly or indirectly) transactions B, D, F, G, so the score of A is $1 + 3 + 1 + 3 + 1 = 9$. Analogously, the score of C is $1 + 1 + 1 + 3 + 1 = 7$.

3 Stability of the system, and cutsets

Let $L(t)$ be the total number of tips (i.e., unapproved transactions) in the system at time t. One, of course, expects that the stochastic process $L(t)$ remains *stable* (more precisely, *positive recurrent*). Intuitively, $L(t)$ should fluctuate around a constant value, and not escape to infinity (thus leaving a lot of unapproved transactions behind).

To analyze the stability properties of $L(t)$, we need some assumptions. Let λ be the rate of the input (Poisson) flow of transactions; for simplicity, let us assume

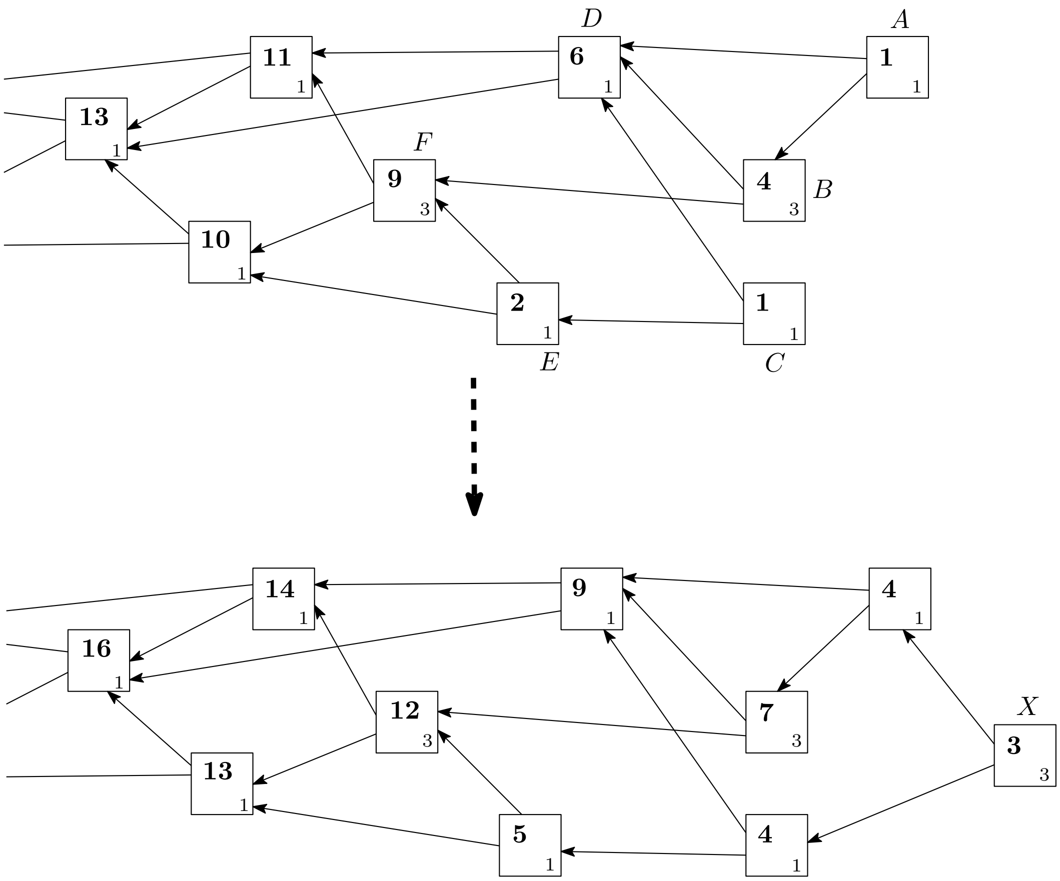

Figure 1: On the weights (re)calculation

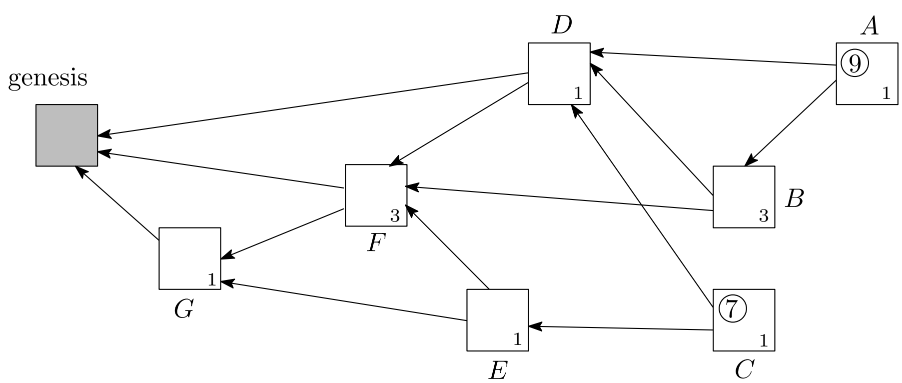

Figure 2: On the calculation of scores (circled)

for now that it remains constant. Assume that all devices have approximately the same computing power, and let $h(L, N)$ be the average time a device needs to do the calculations necessary to issue a transaction in the situation when there are L tips and the total number of transactions is N. First, we consider the strategy when, to issue a transaction, a node just chooses two tips at random and approves them. It should be observed that this strategy has a number of practical disadvantages, in particular, it does not offer enough protection against "lazy" or malicious nodes (see Section 4.1 below). On the other hand, we still consider it since it is simpler to analyse, and therefore one may get insight on the system's behavior for more complicated tip selection strategies. For this strategy, one may assume that the Poisson flows of approvals to different tips are independent, and have rate λ/L (this follows e.g. from Proposition 5.2 of [7]). Therefore,

$$\mathbb{P}\big[\text{nobody approves a given tip during time } h(L,N)\big] = \exp\Big(-\frac{\lambda h(L,N)}{L}\Big). \quad (1)$$

This means that the expected increment of the total number of tips at the moment when our device issues a transaction equals to

$$1 - 2\exp\Big(-\frac{\lambda h(L,N)}{L}\Big) \quad (2)$$

(in the above formula, "1" corresponds to the new tip created by the transaction, and the second term is the expected number of "erased" tips). Now, $L(t)$ is in fact a continuous-time random walk on $\mathbb{N} = \{1, 2, 3, \ldots\}$, with nearest-neighbor transitions. Indeed, if the two chosen transactions were already approved by others, then the process jumps one unit to the left, if both chosen transactions were not approved, then the process jumps one unit to the right, and in the last possible case it remain on the same place.

Now, to understand the typical behavior of the process, observe that the drift in (2) is positive for small L and negative (at least in the case when $h(L, N) = o(L)$ as $L \to \infty$; or just assuming that the main contribution to the computation/propagation time does not come from handling the tips) for large L. The "typical" value of L would be where (2) vanishes, that is, L_0 such that

$$L_0 \approx \frac{\lambda h(L_0, N)}{\ln 2} \approx 1.44 \cdot \lambda h(L_0, N). \quad (3)$$

Clearly, L_0 defined above is also the typical size of the set of tips. Also, the expected time for a transaction to be approved for the first time is around L_0/λ.

Also, observe that (at least in the case when the nodes *try* to approve tips) at any fixed time t the set of transactions that were tips at some moment $s \in [t, t+h(L_0, N)]$ typically constitutes a *cutset*, in the sense that any path from a transaction issued at time $t' > t$ to the genesis must pass through this set. It is important that the size of the cutsets becomes small at least occasionally; one may then use the small cutsets as checkpoints, for possible DAG pruning and other tasks.

It is important to observe, however, that the above "purely random" approval strategy is not very good in practice, because it does not encourage approving tips: a "lazy" user could just always approve a fixed couple of very old transactions (and therefore not contributing to approval of more recent transactions) without being punished for such behavior. Also, a malicious entity can artificially inflate the number of tips (e.g., by issuing a lot of transactions that approve a fixed pair of transactions), in order to make the new transactions select those tips with very high probability, effectively abandoning the tips belonging to "honest" nodes. To avoid issues of this sort, one has to adopt a strategy which is biased towards the "better" tips. One example of such a strategy is presented in Section 4.1 below[2].

As for the expected time for a transaction to be approved for the first time, the situation a bit more complicated. We can distinguish essentially two regimes (see Figure 3):

- low load: the flow of transactions is slow enough, so that, even in the case when the number of tips is small, it is not probable that several different transactions approve the same tip;

- high load: the flow of transactions is big enough, so that typically the number of tips remains large.

In the low load regime, the situation is relatively simple: the first approval happens in average time of order λ^{-1}, since already the first (or one of the first) incoming transactions will approve our transaction.

Let us now consider the high load regime. First, if the transaction did not make it into the top αL, then this waiting time can be quite large, of order $\exp(cL_0^{(\alpha)})$ (since there is a drift towards $L_0^{(\alpha)}$ for smaller values of L and the size of the tip set needs to become much smaller than $L_0^{(\alpha)}$ for that transaction to be considered for approval).

[2]In fact, the author's feeling is that the tip approval strategy is *the* most important ingredient for constructing a tangle-based cryptocurrency. It is there that many attack vectors are hiding. Also, since there is usually no way to *enforce* a particular tip approval strategy, it must be such that the nodes would voluntarily choose to follow it knowing that at least a good proportion of other nodes does so.

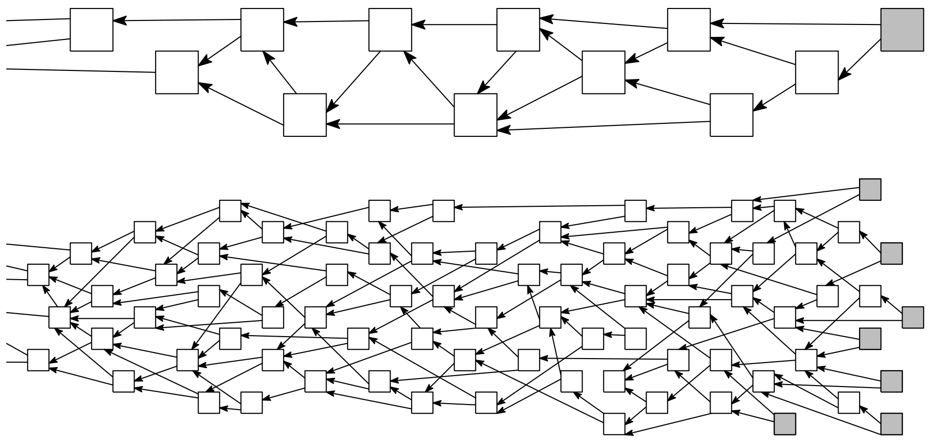

Figure 3: The tangle and its typical tip sets (shaded) in low load and high load regimes. Observe that in the latter case some transactions may have to wait a lot until approved for the first time.

Therefore, a good strategy for the owner of such a transaction would be to issue an additional empty transaction referencing the first one, and hope that this new transaction enters into the top αL. Also, similarly to the above, if the transaction is one of the top αL, then with constant probability it will have to wait around L_0/λ time units to be approved for the first time (observe that $\alpha L_0^{(\alpha)} = L_0$). However, if that did not happen, then the transaction may fall below the top αL, and then a good strategy will be to promote it with an additional empty transaction.

Now, it turns out that the approval strategies based on heights and scores may be vulnerable to a specific type of attacks, see Section 4.1. We will discuss more elaborated strategies to defend against such attacks in that section. Nevertheless, we stress that it is still worth considering the simplest tip selection strategy ("approve two random tips"), since it is the easiest to analyse, and therefore may give some insight about the qualitative and quantitative behavior of the system.

Conclusions:

1. We distinguish between two regimes, low load and high load, as depicted on Figure 3.

2. In the low load regime, usually there are not many tips (say, one or two), and a tip gets approved for the first time in $\Theta(\lambda^{-1})$, where λ is the rate of the incoming flow of transactions.

3. In the high load regime, the typical number of tips depends on the approval strategy (i.e., how the new transaction chooses the other two transactions for approval).

4. For the "approve two random tips" strategy, the typical number of tips is given by (3). It can be shown that this strategy is optimal with respect to the typical number of tips; however, it is not practical to adopt it because it does not encourage approving tips.

5. however, more elaborated strategies are needed; a family of such strategies will be described in Section 4.1.

6. In the high load regime, the typical time until a tip is approved is $\Theta(h)$, where h is the average computation/propagation time for a node. However, if the first approval did not occur in the above time interval, it is a good idea (for the issuer/receiver) to promote that transaction with an additional empty transaction(s).

3.1 How fast does the cumulative weight typically grow?

In the low load regime, after our transaction gets approved several times, its cumulative weight will grow with speed λw, where w is the average weight of a generic transaction, since essentially all new transactions will indirectly reference our transaction.

As for the high load regime, as observed earlier in this section, if a transaction is old enough and with big cumulative weight, then the cumulative weight grows with speed λw for the same reasons. Also, we saw that in the beginning the transaction may have to wait for some time to be approved, and it is clear that its cumulative weight behaves in a random fashion at that time. To see, how fast does the cumulative weight grow after the transaction gets several approvals, let use denote (for simplicity, we start counting time at the moment when our transaction has been created) by $H(t)$ the expected cumulative weight at time t, and by $K(t)$ the expected number of tips that approve our transaction at time t (or simply "our tips"). Let us also abbreviate $h := h(L_0, N)$. Also, we make a simplifying assumption that the overall number of tips remains roughly constant (equal to L_0) in time. We work with the "approve two random tips" strategy here; it is expected that the result will be roughly the same for the strategy "approve two random tips in top $\alpha L(t)$".

Observe that a transaction coming to the system at time t typically chooses the two transactions to approve based on the state of the system at time $t - h$. It is not difficult to obtain that the probability that it approves at least one "our" tip is $\frac{K(t-h)}{L_0}\big(2-\frac{K(t-h)}{L_0}\big)$. We can then write the following differential equation (analogously e.g. to Example 6.4 of [7]):

$$\frac{dH(t)}{dt} = w\lambda \frac{K(t-h)}{L_0}\Big(2-\frac{K(t-h)}{L_0}\Big). \tag{4}$$

In order to be able to use (4), we need first to calculate $K(t)$. It is not immediately clear how to do this, since a tip at time $t-h$ may not already be a tip at time t, and, in the case when the newly coming transaction approves such a tip, the overall number of tips approving the original transaction increases by 1. Now, the crucial observation is that the probability that a tip at time $t-h$ remains a tip at time t is approximately $1/2$, recall (1) and (3). So, at time t essentially a half of $K(t-h)$ "previous" our tips remain to be tips, while the other half will be already approved at least once. Let us denote by A the set of those (approximately) $K(t-h)/2$ tips at time $t-h$ that remained tips at time t, and the set of other $K(t-h)/2$ tips (that were already approved) will be denoted by B. Let p_1 be the probability that the newly arrived transaction approves at least 1 transaction from B and does not approve any transactions from A. Also, let p_2 be the probability that both approved transactions belong to A. Clearly, p_1 and p_2 are, respectively, the probabilities that the current number of "our" tips increases or decreases by 1 upon arrival of the new transaction. Again, some elementary considerations show that

$$\begin{aligned} p_1 &= \frac{K(t-h)}{L_0}\Big(1-\frac{K(t-h)}{2L_0}\Big) - \Big(\frac{K(t-h)}{2L_0}\Big)^2, \\ p_2 &= \Big(\frac{K(t-h)}{2L_0}\Big)^2. \end{aligned}$$

Analogously to (4), the differential equation for $K(t)$ then writes:

$$\frac{dK(t)}{dt} = (p_1-p_2)\lambda = \lambda\frac{K(t-h)}{L_0}\Big(1-\frac{K(t-h)}{L_0}\Big). \tag{5}$$

It is difficult to solve (5) exactly, so we make further simplifying assumptions. First of all, we observe that, after the time when $K(t)$ reaches level εL_0 for a fixed $\varepsilon > 0$, it will grow very quickly almost to L_0. Now, when $K(t)$ is small with respect to L_0, we can drop the last factor in the right-hand side of (5). Also, substituting $K(t-h)$ by $K(t) - h\frac{dK(t)}{dt}$, we obtain a simplified version of (5) (recall that $\frac{\lambda h}{L_0} = \ln 2$):

$$\frac{dK(t)}{dt} \approx \frac{\lambda}{1+\ln 2}\frac{K(t)}{L_0} \approx 0.59\cdot\frac{\lambda K(t)}{L_0}, \tag{6}$$

with boundary condition $K(0) = 1$. This differential equation solves to

$$K(t) \approx \exp\Big(\frac{t \ln 2}{(1+\ln 2)h}\Big) \approx \exp\Big(0.41\frac{t}{h}\Big). \tag{7}$$

So, taking logarithms in (7), we find that the time when $K(t)$ reaches εL_0 is roughly

$$t_0 \approx \big(1 + (\ln 2)^{-1}\big)h \times \big(\ln L_0 - \ln \varepsilon^{-1}\big) \lesssim 2.44 \cdot h \ln L_0. \tag{8}$$

Turning back to (4) (and, as before, dropping the last term in the right-hand side) we obtain that during the "adaptation period" (i.e., $t \le t_0$ with t_0 as in (8)), it holds that

$$\begin{aligned}\frac{dH(t)}{dt} &\approx \frac{2w\lambda}{L_0 \exp\big(\frac{\ln 2}{1+\ln 2}\big)} K(t) \\ &\approx \frac{2w\lambda}{L_0 \exp\big(\frac{\ln 2}{1+\ln 2}\big)} \exp\Big(\frac{t\ln 2}{(1+\ln 2)h}\Big),\end{aligned}$$

and so

$$H(t) \approx \frac{2(1+\ln 2)w}{\exp\big(\frac{\ln 2}{1+\ln 2}\big)} \exp\Big(\frac{t \ln 2}{(1+\ln 2)h}\Big) \approx 2.25 \cdot w \exp\Big(0.41\frac{t}{h}\Big). \tag{9}$$

Let us remind the reader that, as discussed above, after the adaptation period the cumulative weight $H(t)$ grows essentially linearly with speed λw. We stress that the "exponential growth" (as in (9)) does not mean that the cumulative weight grows "very quickly" during the adaptation period; rather, the behavior is as shown on Figure 4.

Also, we comment that the calculations in this section can be easily adapted to the situation when a node references $s > 1$ transactions *in average*. For this, one just need to replace 2 by s in (2) (but not in (4)!), and $\ln 2$ by $\ln s$ in (3) and in (6)–(9).

Conclusions:

1. In the low load regime, after our transaction gets approved several of times, its cumulative weight will grow with speed λw, where w is the mean weight of a generic transaction.

2. In the high load regime, again, after our transaction gets approved several of times, first its cumulative weight $H(t)$ grows with increasing speed during

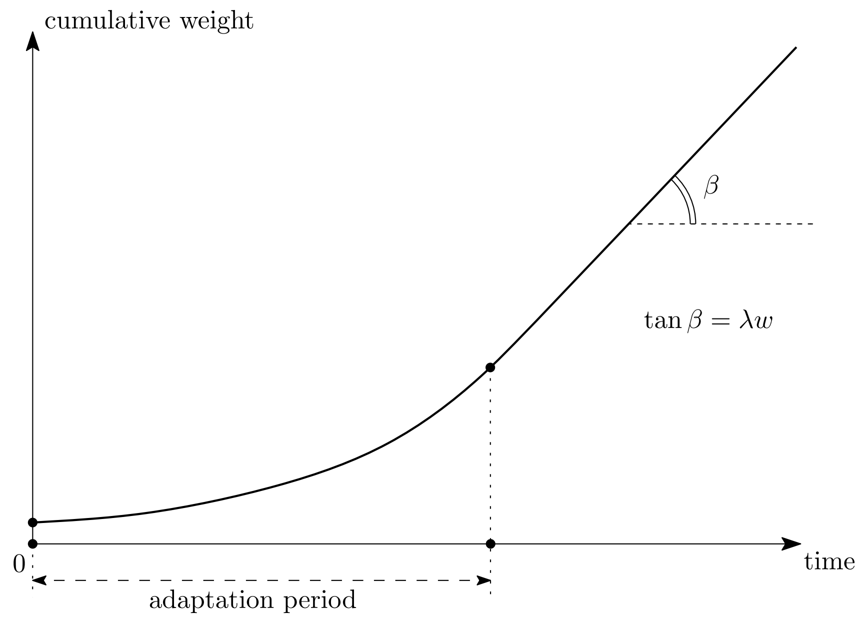

Figure 4: On the cumulative weight growth

the so-called *adaptation period* according to the formula (9), and after the adaptation period is over, it grows with speed λw, see Figure 4. In fact, for *any* reasonable strategy the cumulative weight will grow with this speed after the end of the adaptation period, because essentially all newly coming transactions will indirectly approve our transaction.

3. One can think of the adaptation period as the time until most of the current tips (indirectly) approve our transaction. The typical length of the adaptation period is given by (8).

4 Possible attack scenarios

We start by discussing an obvious attack scenario, where the attacker is trying to "outpace" the network alone:

1. the attacker pays to the merchant, and receives the goods after the merchant considers that the transaction got already a sufficiently large cumulative weight;

2. the attacker issues a double-spending transaction;

3. the attacker issues a lot of small transactions (very aggressively, with all his computing power) that do not approve the original one directly or indirectly, but approve the double-spending transaction;

4. observe that the attacker may have a lot of Sybil identities, and also is not required to necessarily approve tips;

5. (s)he hopes that his/her "sub-DAG" outpaces the main one, so that the DAG continues growing from the double-spending transaction, and the legitimate branch is discarded.

It is possible to argue that arbitrarily large weights should not be allowed; see Section 4 of [6] for an explanation. Therefore, to simplify, we assume that the weights of all transactions are *equal* to w (to be consistent with the previously used notation), and estimate the probability that the attack succeeds. Assume that a given transaction gained cumulative weight w_0 in t_0 time units after the moment when it was issued. Assume also that the adaptation period for that transaction is over, and so its cumulative weight increases linearly with speed λw. Now, the attacker wants to double-spend on this transaction; for that, at the time[3] when the first transaction was issued, he secretly prepares the double-spending transaction, and starts generating other transactions that approve the double-spending one. If at some moment (after the merchant decides to accept the legitimate transaction) the attacker's subtangle outpaces the legitimate subtangle, then the double-spending attack would be successful. If that does not happen, then the double-spending transaction will not be approved by others (because the legitimate transaction will acquire more cumulative weight and essentially all new tips would indirectly approve it), and so it will be orphaned.

As before, let μ stand for the computing power of the attacker (i.e., the speed of weight generation). Let $G_1, G_2, G_3, \ldots$ denote i.i.d. Exponential random variables with parameter μ/w (i.e., with expected value w/μ), and denote also $V_k = \frac{\mu}{w} G_k$, $k \geq 1$. Clearly, $V_1, V_2, V_3, \ldots$ are i.i.d. Exponential random variables with parameter 1.

Suppose that at time t_0 the merchant decided to accept the transaction (recall that it has cumulative weight w_0 at that time). Let us estimate the probability that the attacker successfully double-spends. Let $M(\theta) = (1-\theta)^{-1}$ be the moment generating function (see Section 7.7 of [9]) of the Exponential distribution with parameter 1. It is known (besides the general book [8], see also Proposition 5.2 in Section 8.5 of [9], even though it does not explain why the inequality should be, in fact, an approximate

[3]or even before; we discuss this case later

equality) that for $\alpha \in (0,1)$ it holds that

$$\mathbb{P}\Big[\sum_{k=1}^{n} V_k \leq \alpha n\Big] \approx \exp\big(-n\varphi(\alpha)\big), \tag{10}$$

where $\varphi(\alpha) = -\ln\alpha + \alpha - 1$ is the Legendre transform of $\ln M(-\theta)$. Observe that, as a general fact, it holds that $\varphi(\alpha) > 0$ for $\alpha \in (0,1)$ (recall that the expectation of an Exp(1) random variable equals 1).

Assume also that $\frac{\mu t_0}{w_0} < 1$ (otherwise, the probability that the attacker's subtangle eventually outpaces the legitimate one would be close to 1). Now, to outweight w_0 at time t_0, the attacker needs to be able to issue at least w_0/w (for simplicity, we drop the integer parts) transactions with maximal weight m during time t_0. Therefore, using (10), we obtain that the probability that the double-spending transaction has more cumulative weight at time t_0 is roughly

$$\begin{aligned}
\mathbb{P}\Big[\sum_{k=1}^{w_0/m} G_k < t_0\Big] &= \mathbb{P}\Big[\sum_{k=1}^{w_0/w} V_k < \frac{\mu t_0}{w}\Big] \\
&= \mathbb{P}\Big[\sum_{k=1}^{w_0/w} V_k < \frac{w_0}{w} \times \frac{\mu t_0}{w_0}\Big] \\
&\approx \exp\Big(-\frac{w_0}{w}\varphi\big(\tfrac{\mu t_0}{w_0}\big)\Big).
\end{aligned} \tag{11}$$

That is, for the above probability to be small, we typically need that $\frac{w_0}{m}$ is large and $\varphi\big(\frac{\mu t_0}{w_0}\big)$ is not very small.

Analogously, the probability that the double-spending transaction has more cumulative weight at time $t \geq t_0$ is roughly

$$\exp\Big(-\frac{w_0 + w\lambda(t-t_0)}{w}\varphi\big(\tfrac{\mu t}{w_0 + w\lambda(t-t_0)}\big)\Big). \tag{12}$$

Observe that, typically, we have $\frac{\mu t_0}{w_0} \geq \frac{\mu}{w\lambda}$ (since, during the adaptation period, the cumulative weight grows with speed less than λw). Anyhow, it can be shown that the probability of achieving a successful double spend is of order

$$\exp\Big(-\frac{w_0}{w}\varphi\big(\tfrac{\mu t_0}{w_0} \vee \tfrac{\mu}{w\lambda}\big)\Big), \tag{13}$$

where $a \vee b := \max(a,b)$. For example, let $w = 1$, $\mu = 2$, $\lambda = 3$ (so that the attacker's power is only a bit less that that of the rest of the network). Assume that the transaction got cumulative weight 32 by time 12. Then, $\frac{\mu t_0}{w_0} \vee \frac{\mu}{w\lambda} = \frac{3}{4}$, $\varphi\big(\frac{3}{4}\big) \approx 0.03768$, and (13) then gives the upper bound approximately 0.29. If we assume, however, that $\mu = 1$ (and keep other parameters intact), then $\frac{\mu t_0}{w_0} \vee \frac{\mu}{w\lambda} = \frac{3}{8}$, $\varphi\big(\frac{3}{8}\big) \approx 0.3558$, and (13) gives approximately 0.00001135, quite a change indeed.

From the above discussion it is important to observe that, for the system to be secure, it should be true that $\lambda w > \mu$ (otherwise, the estimate (13) would be useless); i.e., the input flow of "honest" transactions should be large enough compared to the attacker's computational power. This indicates the need for additional security measures (i.e., checkpoints) during the early days of a tangle-based system.

Also, as for the strategies for deciding which one of two conflicting transactions is valid, one has to be careful when relying only on the cumulative weight. This is because it can be subject to an attack similar to the one described in Section 4.1 (the attacker may prepare a double-spending transaction well in advance, build a secret subchain/subtangle referencing it, then broadcast that subtangle after the merchant accepts the legitimate transaction). Rather, a better method for deciding between two conflicting transactions might be the one described in the next section: run the tip selection algorithm, and see which transaction of the two is (indirectly) approved by the selected tip.

4.1 A parasite chain attack and a new tip selection algorithm

Consider the following attack (Figure 5): an attacker secretly builds a chain/subtangle, occasionally referencing the main tangle to gain more score (note that the score of good tips is roughly the sum of all own weights in the main tangle, while the score of the attacker's tips also contains the sum of all own weights in the parasite chain). Also, since the network latency is not an issue to someone with a sufficiently strong computer who builds the chain alone, the attacker might be able to give more height to the parasite tips. Finally, the number of attacker's tips can be artificially increased at the moment of the attack, in case the honest nodes use some selection strategy that involves a simple choice between available tips.

To defend against this attack, we are going to use the fact that the main tangle is supposed to have more (active) hashing power, and therefore manages to give more cumulative weight to more transactions than the attacker. The idea is to use a MCMC (Markov Chain Monte Carlo) algorithm to select the two tips to reference.

Let $\mathcal{H}_x$ be the current cumulative weight of a site (i.e., a transaction represented on the tangle graph). For simplicity, assume that all own weights are equal to 1; so, the cumulative weight of a tip is always 1, and the cumulative weight of other sites

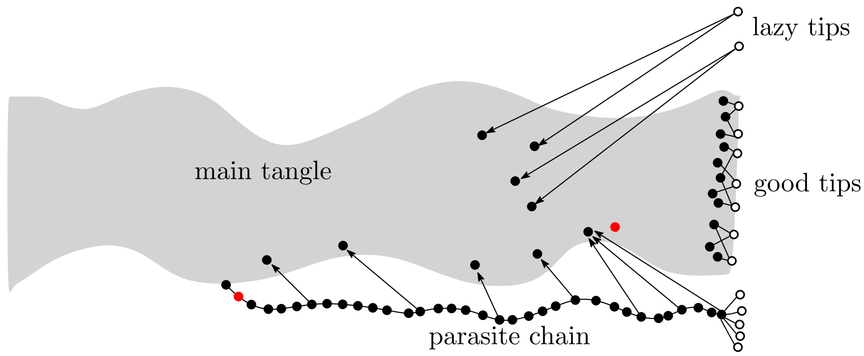

Figure 5: On the tip selection algorithm. The two red circles indicate an attempted double-spend.

is at least 2.

The algorithm is then described in the following way:

1. consider all transactions with cumulative weight between L and (say) $2L$ (where L is large, to be chosen);

2. place N particles independently there (N is not so big, say, 10 or so);

3. these particles will perform discrete-time random walks "towards the tips" (i.e., transition from x to y is possible if and only if y approves x);

4. the two random walks that reach the tip set first will indicate our two tips to approve;

5. the transition probabilities of the walks are defined in the following way: if y approves x (we denote this $y \rightsquigarrow x$), then the transition probability P_{xy} is proportional to $\exp\big(-\alpha(\mathcal{H}_x - \mathcal{H}_y)\big)$, that is

$$P_{xy} = \exp\big(-\alpha(\mathcal{H}_x - \mathcal{H}_y)\big)\Big(\sum_{z:z\rightsquigarrow x} \exp\big(-\alpha(\mathcal{H}_x - \mathcal{H}_z)\big)\Big)^{-1}, \tag{14}$$

   where $\alpha > 0$ is a parameter to be chosen (one can start e.g. with $\alpha = 1$).

In particular, note that this algorithm is "local", one need not go to all the way to the genesis to calculate things.

To see that the algorithm works as intended, consider first the "lazy tips" (those that intentionally approve some old transactions to avoid doing verification), see

Figure 5. Observe that, even if the particle is in a site approved by such a tip, it is not probable that the lazy tip would be selected, since the cumulative weights difference will be very large, look at (14).

Next, consider the following attack: the attacker secretly builds a chain (a "parasite chain") containing a transaction that empties his account to another account under his control (indicated as the leftmost red circle on Figure 5). At some point the attacker issues a transaction in the main tangle (indicated as the other red circle), and waits until the merchant accepts it. The parasite chain occasionally references the main tangle (hence the name) and so its sites have good height/score (even better than those of the main tangle), although the cumulative weight is not so big in that chain. Note also that it cannot reference the main tangle after the merchant's transaction. Also, the attacker might try to artificially inflate the number of "his" tips at the moment of the attack, as shown on the picture. The attacker's idea is to make the nodes reference the parasite chain, so that the "good" tangle would become orphaned.

Now, it is easy to see why the MCMC selection algorithm with high probability will not select one of the attacker's tips. Basically, the reason is the same as why the algorithm doesn't select the lazy tips: the sites of the parasite chain will have a much smaller cumulative weight than the main tangle's sites they reference. Therefore, it is not probable that the random walk will ever jump to the parasite chain (unless it begins there, but this is not very probable too, since the main tangle contains more sites).

Also, it is not set in stone that the transition probabilities should necessarily be defined as in (14). Instead of the exponent, one can take some other rapidly decreasing function, such as e.g. $f(s) = s^{-3}$.

4.2 Splitting attack

The following attack scheme against the above MCMC algorithm was suggested by Aviv Zohar. In the high-load regime, an attacker can try to split the tangle in two branches and maintain the balance between them, so that both continue to grow. To avoid that a honest node references the two branches at once (effectively joining them), the attacker must place at least two conflicting transactions in the beginning of the split. Then, (s)he hopes that roughly half of the network would contribute to each branch, so that (s)he would be able to "compensate" random fluctuations even with a relatively small computing power. Then, the attacker would be able to spend the same funds on the two branches.

To defend against such an attack, one needs to use some "sharp-threshold" rule (like "select the longest chain" in Bitcoin) that makes it too hard to maintain the balance between the two branches. Just to give an example, assume that one branch has the total weight (or any other metric that we may use) 537, and the total weight of the other branch is quite close, say, 528. If in such a situation a honest node selects the first branch with probability very close to 1/2, then, probably, the attacker would be able to maintain the balance between the branches. If, however, a honest node prefers the first branch with probability considerably bigger than 1/2, then the attacker would probably be unable to maintain the balance, because after an inevitable random fluctuation the network will quickly choose one of the branches and abandon the other. Clearly, to make the MCMC algorithm behave this way, one has to choose a very rapidly decaying function f, and also start the random walk at a node with (relatively) large depth (so that it is probable that it starts before the split was created). In this case the random walk would choose the "heavier" branch with big probability, even if the total weight difference between the branches is small.

One may consider other modifications of the tip selection algorithm. For example, if a node sees two big subtangles, then it first chooses the one with larger sum of own weights, and then does the tip selection only there using the above MCMC algorithm.

Also, the following idea may be worth considering: make the transition probabilities in (14) depend not only on $\mathcal{H}_x - \mathcal{H}_y$, but on $\mathcal{H}_x$ too, in such a way that the next step of the Markov chain is almost deterministic when the walker is deep in the tangle (to avoid entering the weaker branch), but becomes more spread out when close to tips (so that there is enough randomness in the choice of the two transactions to approve).

Conclusions:

1. We considered a simple attack strategy, when the attacker tries to double-spend by "outpacing" the system. When the input flow of "honest" transactions is large enough compared to the attacker's computational power, the probability that the double-spending transaction has more cumulative weight can be estimated using the formula (13) (see also examples below (13)).

2. The attack based on building a "parasite chain" makes the approval strategies based on height or score obsolete, since the attacker will get higher values of those than the legitimate tangle. On the other hand, the MCMC tip selection algorithm described in Section 4.1 seems to protect well against this kind of attack.

3. As a bonus, it also offers protection against the "lazy nodes", i.e., those that just approve some old transactions to avoid doing the calculations necessary for validating the tangle.

References

[1] Iota: a cryptocurrency for Internet-of-Things. See http://www.iotatoken.com/ and https://bitcointalk.org/index.php?topic=1216479.0

[2] PEOPLE ON NXTFORUM.ORG (2014) DAG, a generalized blockchain. https://nxtforum.org/proof-of-stake-algorithm/dag-a-generalized-blockchain/ (registration at nxtforum.org required)

[3] SERGIO DEMIAN LERNER (2015) DagCoin: a cryptocurrency without blocks. https://bitslog.wordpress.com/2015/09/11/dagcoin/

[4] YONATAN SOMPOLINSKY, AVIV ZOHAR (2013) Accelerating Bitcoin's Transaction Processing. Fast Money Grows on Trees, Not Chains. https://eprint.iacr.org/2013/881.pdf

[5] YOAD LEWENBERG, YONATAN SOMPOLINSKY, AVIV ZOHAR (2015) Breaking free from chains: "Secure chainless" protocols for Bitcoin. https://dl.dropboxusercontent.com/u/7426164/Bitcoin/Bitcoin__meetup_Chainless.pptx

[6] SERGUEI POPOV (2015) The tangle (the iota whitepaper). http://188.138.57.93/tangle.pdf

[7] SHELDON M. ROSS (2012) *Introduction to Probability Models.* 10th ed.

[8] AMIR DEMBO, OFER ZEITOUNI (2010) *Large Deviations Techniques and Applications.* Springer.

[9] SHELDON M. ROSS (2009) *A First Course in Probability.* 8th ed.