The tangle

Serguei Popov[*], for `Jinn Labs`

June 17, 2017. Version 1.0

Abstract

In this paper we analyze the technology used as a backbone of iota (a cryptocurrency for Internet-of-Things industry). Its main feature is that, instead of a blockchain, we use a *tangle* (that is, a directed acyclic graph) for storing transactions. This technology naturally succeeds the blockchain technology as its next evolutionary step and comes out with features required for micropayments conducted on a global scale.

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 and description of the system

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. Specifically, in the currently available systems one must pay a fee for making a transaction; so, transferring a *very* small amount just makes no sense since one would have also to pay the fee which is many times larger. On the other hand, it is not easy to get rid of the fees since they serve as an incentive for the creators of the blocks. It should be also observed that existing cryptocurrencies are heterogeneous systems with clear separation of roles (transaction issuers, transaction approvers).

[*]a.k.a. `mthcl`; author's contact information: `e.monetki@gmail.com`

Such systems create unavoidable discrimination of some of their elements which in turn creates conflicts and makes all elements spend resources on conflict resolution. All 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 *iota* [1], recently designed as a cryptocurrency for the Internet-of-Things industry. We, however, focus more on general features and problems/difficulties that arise when 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. Instead of the global blockchain, there is a DAG (= directed acyclic graph) that we call *tangle*. The transactions issued by nodes constitute the site set of the tangle (i.e., the tangle graph is the ledger for storing transactions). 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 no directed edge between transaction A and transaction B but there is a directed path of length at least two from A to B, we say that A *indirectly approves* B. There is also the "genesis" transaction, which is approved (directly or indirectly) by all other transactions, see Figure 2. The genesis is described in the following way. In the beginning there was an address with balance containing all the tokens. Then the genesis transaction sent these tokens to several other "founder" addresses. Let us stress that all the tokens were created in the genesis (no other tokens will be created), and there no mining in the sense "miners receive monetary rewards".

A quick note on the terminology: *sites* are transactions represented on the tangle graph. The network is composed by *nodes*; that is, nodes are entities that issue transactions.

Now, the main idea is the following: to issue a transaction, users must work to approve other transactions, therefore contributing to the network's security. It is assumed that the nodes check if the approved transactions are not conflicting and do not approve (directly or indirectly) conflicting transactions[2]. 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. It is important to observe

[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

[2] if a node issues a new transaction that approves conflicting transactions, then it risks that others will not approve this new transaction, and so it will fall into oblivion

that we do not *impose* any rule for choosing the transactions to approve; rather, we argue that if a large number of other nodes follow some "reference" rule (which seems to be a reasonable assumption, especially in the context of IoT, where nodes are specialized chips with pre-installed firmware), then for any fixed node it is better to stick to a rule of the same kind[3].

More specifically, to issue a transaction, a node does the following:

- First, it chooses two other transactions to approve (in general, these two transactions may coincide), according to some algorithm.
- It checks if the two transactions are not conflicting and do not approve conflicting transactions.
- For the transaction to be valid, the node must solve a cryptographic puzzle (which may be computationally demanding) similar to those in the Bitcoin mining (e.g., it needs to find a nonce such that the hash of that nonce together with some data from the approved transactions has a particular form, for instance, has at least some fixed number of zeros in front).

It is important to observe that, in general, we have an asynchronous network, so that nodes do not necessarily see the same set of transactions. It should be noted also that the tangle may contain conflicting transactions. The nodes do not have to achieve consensus on which valid[4] transactions have the right to be in the ledger (all of them can be there); but, in case there are conflicting transactions, they need to decide which transactions will become orphaned (that is, eventually not indirectly approved by incoming transactions anymore). The main rule that the nodes use for deciding between two conflicting transactions is the following: a node runs the tip selection algorithm[5] (cf. Section 4.1) many times, and see which transaction of the two is more likely to be (indirectly) approved by the selected tip. For example, if, after 100 runs of the tip selection algorithm, a transaction was selected 97 times, we say that it is confirmed with 97% confidence.

Let us also comment on the following question (cf. [4]): what motivates the nodes to propagate transactions? In fact, in our setup the nodes do not have motivation not to propagate. Every node calculates some statistics, one of which is how many new transactions are received from a neighbor. If one particular node is "too lazy", it

[3] we comment more on this in the end of Section 4.1

[4] i.e., transactions issued according to the protocol

[5] as mentioned above, there is a good reason to assume that other nodes would follow (almost) the same algorithm for tip selection

will be dropped by its neighbors. So, even if a node does not issue transactions (and hence has no direct incentive to share new transactions that approve its own one), it still has incentive to participate.

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, in the unlikely event that the reader is scared by the formulas, (s)he can jump directly to the "conclusions" part in the end of corresponding section.

It should be noted that the ideas about usage of DAGs in the cryptocurrency context were around for some time, see e.g. [3, 6, 7, 9, 12]. Specifically, the work [7] introduces the so-called GHOST protocol, which proposes a modification of the Bitcoin protocol by making the main ledger a tree instead of the blockchain; it is shown that such a modification permits to reduce the confirmation times and improve the overall security of the network. In the paper [9] the authors consider a DAG-based cryptocurrency model; differently from our model, the sites of the DAG are blocks (not individual transactions), the miners compete for transactions' fees, and (as in Bitcoin) new tokens may be created. Also, observe that in the work [6] a solution somewhat similar to ours was proposed, although it does not discuss any particular tip approval strategies. After the first version of this paper was published, several other works in that direction has appeared, e.g. [8]. We mention also another approach [2, 10] that aims to make Bitcoin micro-payments possible by establishing peer-to-peer payment channels.

2 Weights and more

Here, we define the (own) weight of a transaction and related concepts. The weight of a transaction is proportional to the amount of work that the issuing node invested into it; in practice (meaning "in current iota's implementation"), the weight may assume only values 3^n, where n is positive integer and belongs to some nonempty interval of acceptable values[6]. In fact, 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 transactions with "acceptable" weights in a short time period.

[6] this interval should also be finite — see the "large weight attack" in Section 4

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 $9 = 3 + 1 + 3 + 1 + 1$, 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 3 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 $1 + 1 + 1 + 3 + 1 = 7$.

In fact, to understand the arguments in this paper, one may safely assume that all weights are equal to 1. So, *from now, on, we stick to this assumption.* In particular, the cumulative weight now becomes one plus the number of transactions that approve our transaction directly or indirectly. The score now is the number of transactions that are directly or indirectly approved by our transaction.

Also, let us observe that, among the above metrics, the cumulative weight is the most important for us (although heights, depths, and scores will briefly enter to some discussions as well).

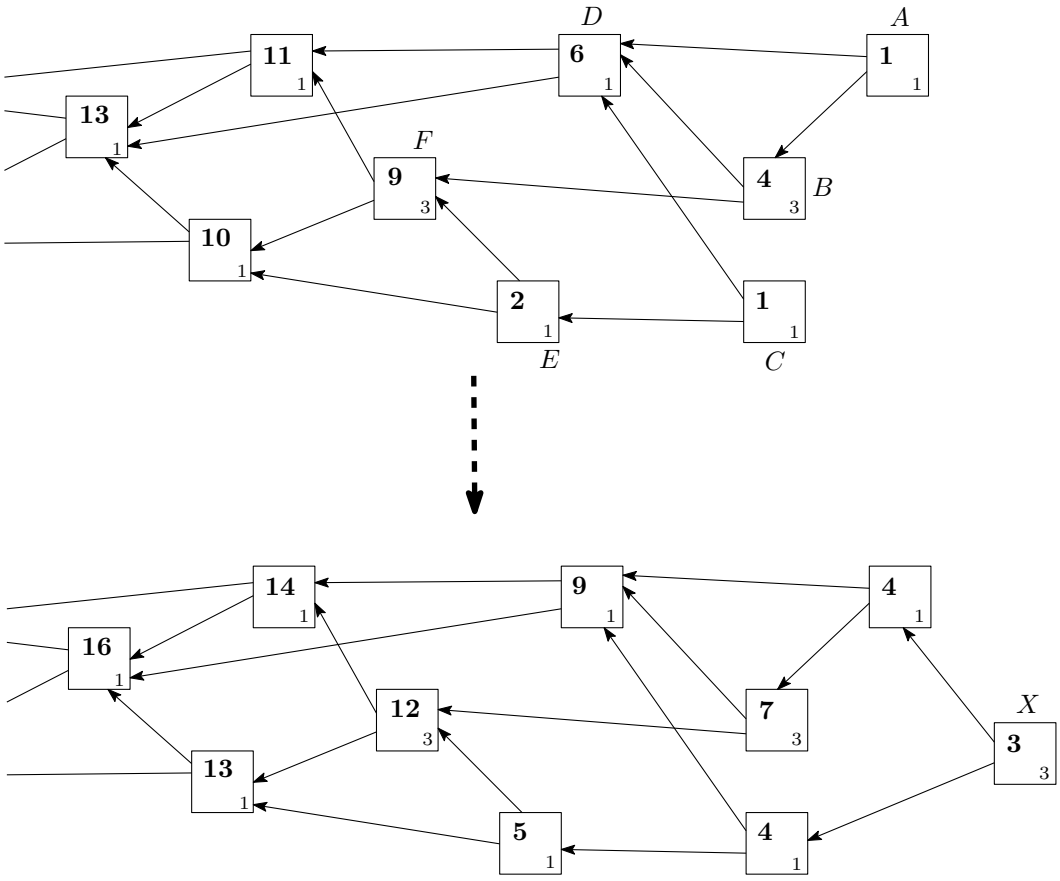

Figure 1: On the weights (re)calculation

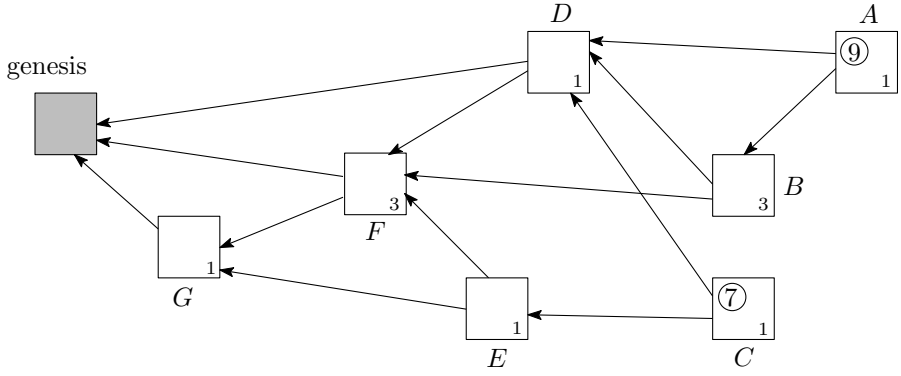

Figure 2: On the calculation of scores (circled)

3 Stability of the system, and cutsets

Let $L(t)$ be the total number of tips (i.e., unreferenced transactions) in the system at time t. One, of course, expects that the stochastic process $L(t)$ remains *stable*[7] (more precisely, *positive recurrent*, see Sections 4.4 and 6.5 of [11] for formal definitions; in particular, positive recurrence implies that the limit of $\mathbb{P}[L(t) = k]$ as $t \to \infty$ should exist and be positive for all $k \geq 1$). Intuitively, we expect that $L(t)$ 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. One of them is that transactions are issued by large number of roughly independent entities, so the process of incoming transactions can be modeled by a Poisson point process (cf. e.g. Section 5.3 of [11]). Let λ be the rate of that Poisson process; for simplicity, let us assume for now that it remains constant in time. 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. Then, let us *assume* that all nodes behave in the following way: to issue a transaction, a node just chooses two tips at random and approves them. It should be observed that, in general, it is *not* a good idea for the "honest nodes" to adopt this strategy. It has a number of practical disadvantages, in particular, it does not offer enough protection against "lazy" or malicious nodes[8] (see Section 4.1 below). On the other hand, we still consider it since it is simpler to analyze, 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 $2\lambda/L$ (this follows e.g. from Proposition 5.3 of [11], which says that if we independently classify each event of a Poisson process according to a list of some possible subtypes, then the processes of events of each subtype are independent Poisson processes). Therefore,

$$\mathbb{P}[\text{nobody approves a given tip during time } h(L,N)] = \exp\Big(-\frac{2\lambda h(L,N)}{L}\Big). \quad (1)$$

This means that the expected increment of the total number of tips at the moment

[7] under an additional assumption that the process is time-homogeneous

[8] we remind the reader that we do not try to *enforce* any particular tip selection strategy, so an attacker can choose tips in any way he/she finds convenient

when our device issues a transaction equals to

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

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. Indeed, according to (1) (and assuming also that the node always chooses two *distinct* tips to approve in the event when there are at least two available tips) each tip initially chosen for approval will still remain tip at the moment when the transaction is issued with probability $\exp\big(-\frac{2\lambda h(L,N)}{L}\big)$. 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 right, if both chosen transactions were not approved, then the process jumps one unit to the left, 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 (i.e., the expectation of the increment of the number of tips) in (2) is positive for small $L(t)$ 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. In other words, if $L(t)$ is small, then it has a tendency to increase, while if $L(t)$ is large, it tends to decrease. Therefore, the "typical" value L_0 of $L(t)$ would be where (2) vanishes (so that the number of tips has no clear tendency to increase or decrease), that is,

$$L_0 \approx \frac{2\lambda h(L_0,N)}{\ln 2} \approx 2.89 \cdot \lambda h(L_0,N). \tag{3}$$

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 $L_0/2\lambda$.

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.

Now, the above "purely random" 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[9]. To discourage the behavior of this sort, one has to adopt a strategy which is biased towards the tips with higher score.

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 (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[10].

Before starting the discussion about the expected time for a transaction to be approved for the first time, observe that we can distinguish essentially two regimes (see Figure 3).

- Low load: the typical number of tips is small, and even frequently becomes 1. This may happen when the flow of transactions is small enough, so that it is not probable that several different transactions approve the same tip; also, if the network latency is very low and the devices compute fast, then, even in the situation when the flow of transaction is reasonably large, it is also unlikely that many tips would appear. Also, we have to assume that there are no attackers that try to inflate the number of tips artificially.

- High load: the typical number of tips is large. This may happen when the flow of transactions is big enough, and the computational delays together with the network latency make it likely that several different transactions approve the same tip.

Of course, this division is rather informal and there is no clear borderline between the the two regimes; nevertheless, we find that it may be instructive to consider these two essentially different situations.

[9] we remind the reader that we do not try to *enforce* any particular tip selection strategy, so an attacker can choose tips in any way he/she finds convenient

[10] 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, that strategy must be such that the nodes would voluntarily choose to follow it knowing that at least a good proportion of other nodes does so.

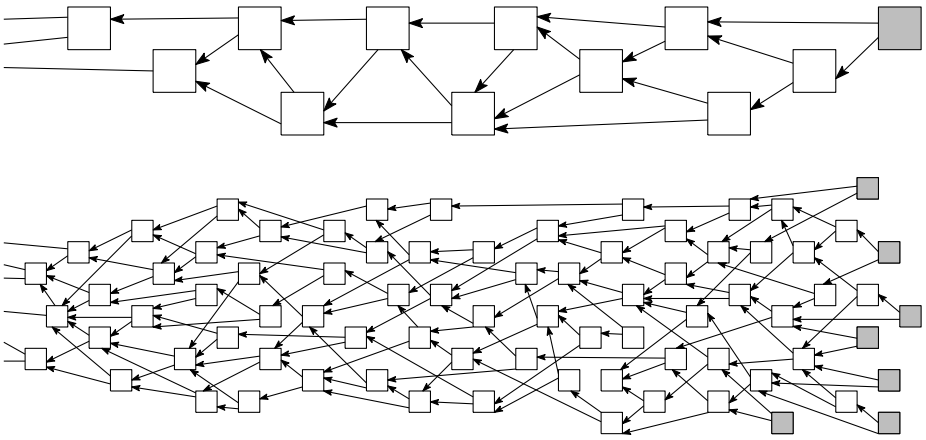

Figure 3: The tangle and its typical tip sets (shaded) in low load and high load regimes. Observe that in the latter case some transactions may have to wait a lot until approved for the first time.

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, that is, the situation when L_0 is large. As mentioned above, one may assume that the Poisson flows of approvals to different tips are independent and have rate approximately λ/L_0. Therefore, the expected time for a transaction to be approved for the first time is around $L_0/(2\lambda) \approx 1.45h$ (recall (3)). It is worth noting, however, that for more elaborated approval strategies (that favor the tips of the "better" quality, whatever it can mean), it may not be a good idea to passively wait a lot of time until the transaction is approved by the others; this is because "better" tips will keep appearing and be preferred for approval. Rather, in case when the transaction is waiting for approval for too long time (i.e., much larger than L_0/λ) a good strategy will be to promote it with an additional empty transaction[11], that is, issue an empty transaction that approves our previous transaction together with one of the "better" tips (so that this empty transaction has a good chance to be further approved).

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

[11] an empty transaction is a transaction that does not involve any token transfer; still, it has to approve two other transactions thus contributing to the network security

elaborated strategies[12] to defend against such attacks in that section. Nevertheless, 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.

[12] 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.

3.1 How fast does the cumulative weight typically grow?

Clearly, in the low load regime, after our transaction gets approved several times, its cumulative weight will grow with speed λ, since essentially all new transactions will indirectly reference our transaction[13].

As for the high load regime, if a transaction is old enough and with big cumulative weight, then the cumulative weight grows with speed λ for the same reasons. Also, we know 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 the cumulative weight grows after the transaction gets several approvals, 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 qualitative behavior will be roughly the same for other reasonable strategies.

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 $t-h$ (because the node must do some calculations/verifications before actually issuing the transaction). It is not difficult to obtain that the probability that it approves at least one "our" tip is $1-\big(1-\frac{K(t-h)}{L_0}\big)^2 = \frac{K(t-h)}{L_0}\big(2-\frac{K(t-h)}{L_0}\big)$ (the expression in the left-hand side is 1 minus probability that the two approved tips are not ours). Analogously e.g. to Example 6.4 of [11], we can write

$$H(t+\delta) = H(t) + \lambda\delta\frac{K(t-h)}{L_0}\Big(2-\frac{K(t-h)}{L_0}\Big) + o(\delta),$$

and thus deduce the following differential equation

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

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 be a tip at time t, and, in the case when the newly coming transaction approves such a tip, the overall number of tips

[13] recall that we assumed that the own weights of all transactions are equal to 1, so the cumulative weight is just the number of all transactions that reference (directly or indirectly) our transaction plus 1

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 $1/2$ (indeed, just plug the expression in (3) to (1)). So, at time t essentially a half of $K(t-h)$ "previous" our tips remain 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 by time t) 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. We have

$$p_1 = \Big(\frac{K(t-h)}{2L_0}\Big)^2 + 2 \times \frac{K(t-h)}{2L_0}\Big(1 - \frac{K(t-h)}{L_0}\Big),$$
$$p_2 = \Big(\frac{K(t-h)}{2L_0}\Big)^2$$

(to obtain the first expression, observe that p_1 equals to the probability that both approved tips belong to B plus twice the probability that the first tip belongs to B and the second tip does not belong to $A \cup B$). 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 to $(1-\varepsilon)L_0$ (indeed, the right-hand side of (5) will be bounded away from 0). Now, when $K(t)$ is small with respect to L_0, we can essentially drop the last factor in the right-hand side of (5) (it will be at least a constant close to 1, so the right-hand side would be equivalent to $\lambda\frac{K(t-h)}{L_0}$). Also, substituting $K(t-h)$ by $K(t) - h\frac{dK(t)}{dt}$, we obtain a simplified version of (5) (recall that $\frac{\lambda h}{L_0} = \frac{1}{2}\ln 2$):

$$\frac{dK(t)}{dt} \approx \frac{\lambda}{1 + \frac{1}{2}\ln 2}\frac{K(t)}{L_0} \approx 0.74 \cdot \frac{\lambda K(t)}{L_0}, \tag{6}$$

with boundary condition $K(0) = 1$. The (approximate) solution of this differential equation is

$$K(t) \approx \exp\Big(\frac{t \ln 2}{(2 + \ln 2)h}\Big) \approx \exp\Big(0.26\frac{t}{h}\Big). \tag{7}$$

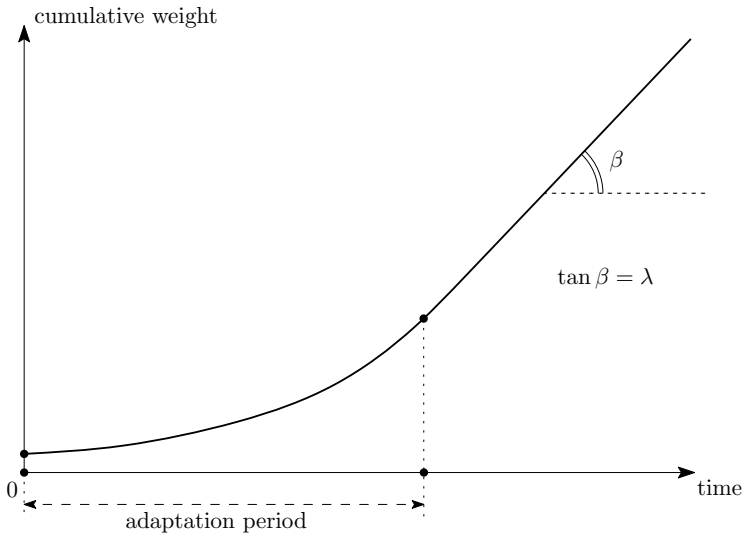

Figure 4: On the cumulative weight growth

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

$$t_0 \approx \big(1 + 2(\ln 2)^{-1}\big)h \times \big(\ln L_0 - \ln \varepsilon^{-1}\big) \lesssim 3.89 \cdot h \ln L_0. \tag{8}$$

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

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

and so

$$H(t) \approx \frac{(2+\ln 2)}{\exp\big(\frac{\ln 2}{2+\ln 2}\big)} \exp\Big(\frac{t \ln 2}{(2+\ln 2)h}\Big) \approx 2.08 \cdot \exp\Big(0.26 \frac{t}{h}\Big). \tag{9}$$

Let us remind the reader that, as discussed above, after the adaptation period the cumulative weight $H(t)$ grows linearly with speed λ. 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 2 by s as well as $\ln 2$ 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 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;

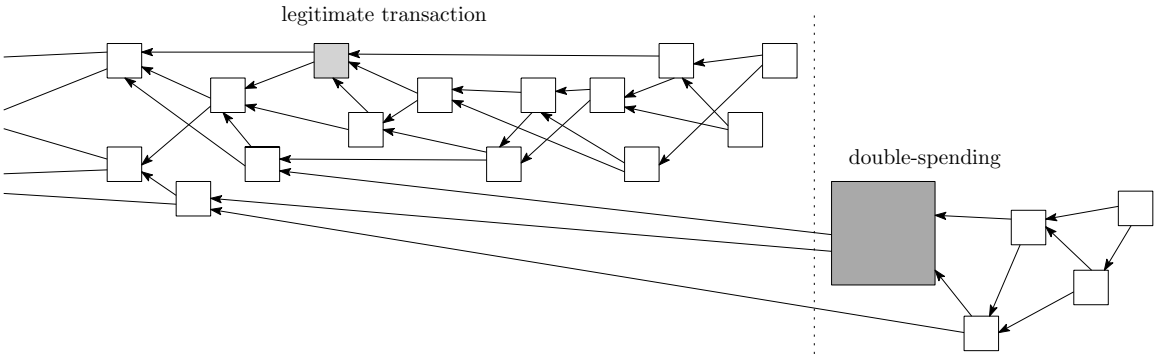

Figure 5: The "large weight" attack

5. alternatively to item 3, the attacker may use all his computing power to issue a "big" double-spending transaction (i.e., with a very large own weight) that approves a couple of transactions prior to the legitimate one (the one used to pay to the merchant);

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

In fact, we will see below that that the strategy of one big double-spending transaction increases the attacker chances. Even more, in the "ideal" situation of this mathematical model, this attack *always* succeeds.

Indeed, let $W^{(n)}$ be the time needed to obtain a nonce which gives weight at least 3^n to the double-spending transaction. One may assume that $W^{(n)}$ is an Exponentially distributed random variable with parameter $\mu 3^{-n}$ (i.e., with expectation $\mu^{-1}3^n$), where μ measures the computing power of the attacker.

Assume that the merchant accepts the legitimate transaction when its cumulative weight becomes at least w_0 (and this happens t_0 time units after the original transaction), and then it is reasonable to expect that the cumulative weight grows with linear speed λw, where λ is the overall arrival rate of transactions to the system (issued by honest users) and w is the mean weight of a generic transaction. Denote $w_1 = \lambda w t_0$, the typical total weight of the legitimate branch at that time.

Being $\lceil x \rceil$ the smallest integer greater than or equal to x, define $n_0 = \left\lceil \frac{\ln w_1}{\ln 3} \right\rceil$, so that $3^{n_0} \geq w_1$ (in fact, $3^{n_0} \approx w_1$ if w_1 is large). If during the time interval of length t_0 the attacker managed to obtain a nonce that gives weight at least 3^{n_0}, then the attack succeeds. The probability of this event is

$$\mathbb{P}[W^{(n_0)} < t_0] = 1 - \exp(-t_0\mu 3^{-n_0}) \approx 1 - \exp(-t_0\mu w_1^{-1}) \approx \frac{t_0\mu}{w_1}$$

(at least in the case when $\frac{t_0\mu}{w_1}$ is small, which is a reasonable assumption). Then, if this "immediate" attack did not succeed, the attacker may continue to look for the nonce that gives weight 3^n for $n > n_0$, and hope that at the moment he finds it, the total weight of the legitimate branch is smaller than 3^n. Now, the probability of this is

$$\mathbb{P}[\lambda w W^{(n)} < 3^n] = 1 - \exp\big(-\mu 3^{-n_0} \times (3^{n_0}/\lambda w)\big) = 1 - \exp(-\mu/\lambda w) \approx \frac{\mu}{\lambda w}.$$

That is, although $\frac{\mu}{\lambda w}$ should be typically small, at each "level" n the attack succeed with a constant probability. Therefore, it will eventually succeed a.s.. In fact, the typical time until it succeeds is roughly $3^{\frac{\lambda w}{\mu}}$. Although this quantity may be very large, still the probability that even the "first" (i.e., during the time t_0) attack succeeds, is not very small. We thus arrive to the following conclusion: we need countermeasures. For example, limit the own weight from above, or even set it to constant (as mentioned in Section 3, the latter may be not the best solution, since it does not offer enough protection from spam).

Now, let us discuss the situation when the maximal weight is capped, say, by 1 (the general case can be treated in the same way, simply by rescaling), 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 λ. Now, the attacker wants to double-spend on this transaction; for that, at the time[14] when the first transaction was issued, he/she 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 transactions' generation). We also make a simplifying assumption that the transactions propagate instantly. Let $G_1, G_2, G_3, \ldots$ denote i.i.d. Exponential random variables with parameter μ (i.e., with expected value $1/\mu$), and denote also $V_k = \mu G_k$,

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

$k \geq 1$. 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 [14]) of the Exponential distribution with parameter 1. It is known[15] 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 Exponential random variable with parameter 1 also 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 outweigh w_0 at time t_0, the attacker needs to be able to issue at least w_0 (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

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

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.

Note that, at time $t \geq t_0$, the cumulative weight of the legitimate transaction is roughly $w_0 + \lambda(t - t_0)$ (recall that we assumed that the adaptation period is over, so the cumulative weight just grows with speed λ). Analogously to (11), one obtains that the probability that the double-spending transaction has more cumulative weight

[15] this is a consequence of the so-called Large Deviation Principle; besides the general book [13], see also Proposition 5.2 in Section 8.5 of [14] for a simple and instructive derivation of the upper bound, and Section 1.9 of [5] for the (not so simple) derivation of the lower bound

at time $t \geq t_0$ is roughly

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

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

$$\exp\big(-w_0\varphi\big(\max(\tfrac{\mu t_0}{w_0}, \tfrac{\mu}{\lambda})\big)\big). \tag{13}$$

For example, let $\mu = 2$, $\lambda = 3$ (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, $\max(\frac{\mu t_0}{w_0}, \frac{\mu}{\lambda}) = \frac{3}{4}$, $\varphi\big(\frac{3}{4}\big) \approx 0.03768$, and (13) then gives the upper bound approximately 0.29. If we assume, however, that $\mu = 1$ (and keep other parameters intact), then $\max(\frac{\mu t_0}{w_0}, \frac{\mu}{\lambda}) = \frac{3}{8}$, $\varphi\big(\frac{3}{8}\big) \approx 0.3558$, 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 $\lambda > \mu$ (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 6): 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[16], 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 (by issuing a lot of transactions that all approve *the same* attacker's transactions, see Figure 6 below), 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). Recall that we assumed 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 is at least 2.

The idea is to place some particles (a.k.a. random walkers) on sites of the tangle, and let them walk towards the tips in a random[17] way. The tips "chosen" by the walks are then the candidates for approval. The algorithm is described in the following way:

1. consider all transactions with cumulative weight between W and (say) $2W$ (where W is reasonably large, to be chosen[18]);

2. place N particles independently there (N is not so big, say, 10 or so[19]);

3. these particles will perform independent 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 (however, it may be wise to modify this rule in the following way: first discard those random walkers that reached the tips *too fast*: they may have ended in one of the "lazy tips");

[16] this is because the attacker can always approve his/her own transactions, without relying on any information from the rest of the network

[17] there is no "canonical" source of randomness; the nodes just use their own (pseudo)random number generators to simulate the random walks

[18] the idea is to place the particle "deep" into the tangle (so that it will not come to a tip straight away) but not "too deep" (so that it will find a tip in a reasonable time). Also, the $[W, 2W]$ interval is arbitrary, one could chose e.g. $[W, 3W]$ instead, or whatever. Other ways of selecting the walkers' starting points are also possible; for example, a node can simply take a random transaction received between e.g. t_0 and $2t_0$ units of time ago, where t_0 is some fixed time value.

[19] again, this choice is largely arbitrary; we use several particles instead of just two for additional security. The idea is that a particle accidentally jumps to the attacker's chain (which is supposed to be long), then it will spend a lot of time there and other tips will be chosen first.

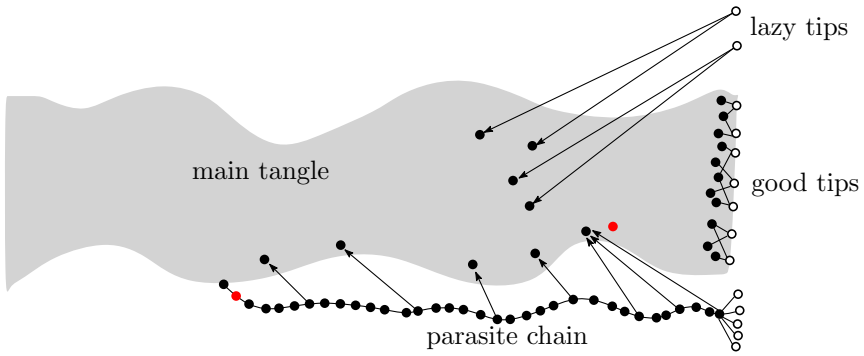

Figure 6: On the tip selection algorithm. The two red circles indicate an attempted double-spend.

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

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

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

Note that this algorithm is "local", one need not go to all the way to the genesis to calculate things. In particular, observe that one does not need to calculate the cumulative weights for the whole tangle; at most one needs to do it for the sites that (indirectly) approve the starting point of the walk.

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 6. 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/her account to another account under his/her control (indicated as the leftmost red circle on Figure 6). 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/her 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 does not 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).

Next, let us comment on why the nodes would follow (at least approximately) this algorithm. Recall that, as observed in Section 1, it is reasonable to assume that at least a "good" proportion of the nodes will follow the *reference* algorithm. Also, because of computational and network delays, the tip selection algorithm would rather work with a past snapshot of the tangle (with respect to the moment when the transaction is issued). Now, *it may be a good idea to intentionally move this snapshot more to the past*[20] in the reference algorithm, for the reasons that we explain in the sequel. Imagine a "selfish" node that just wants to maximize the chances that his/her transaction will be soon approved. The MCMC algorithm of this section (adopted by a considerable proportion of other nodes) defines a probability distribution on the set of tips; clearly, a natural first choice for that node would be picking the tips where the maximum of that distribution is attained. However, if many other nodes also behave in a selfish way (which is quite reasonable to assume as well) and use the same strategy, then they all will lose: *many* new transactions will approve the same two tips at (roughly) the same time, so there will be too much competition between them for subsequent approval (since the nodes use a past snapshot, they will not yet "feel" the cumulative weight increase caused by this mass approval of the two tips). So, even a selfish node would have to use some random tip approval algorithm[21], and the probability distribution of the selected tips should be, in some

[20] that is, first the random walk finds a (past) tip with respect to that snapshot, and then it continues to walk towards the "actual" tips on the current tangle

[21] there seem to be no easy way to discover which tips are better (that is, more likely to be selected by "honest" nodes) other than running the MCMC many times. However, running MCRW many times requires time and other resources; after one spends some time on it, the state of the tangle will already change, so one would possibly even have to start anew. This explains why nodes do not have reasons to abandon the MCMC tips selection strategy in favor of something else, at least

sense, "not far away" from the default probability distribution (i.e., the one produced by the reference tip selection algorithm). We do not claim that this "aggregated" probability distribution (in the presence of selfish nodes) would be equal to the default probability distribution, but the above argument shows that it should be close to it. This means that the probability of selecting "bad" tips would remain small. In any case (differently from Bitcoin), there is not so much incentive for the nodes to be selfish, since possible gains amount only to slight decrease in confirmation time. The nodes do not have reasons to abandon the MCMC tips selection

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. $f(s) = s^{-3}$. There is much freedom for the choice of W and N as well. In fact, the author's feeling is that the main contribution of this section is the very idea of using MCMC for tip selection; it is unclear if there is any "theoretical" argument that shows exactly in which way these parameters must be chosen.

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, he/she hopes that roughly half of the network would contribute to each branch, so that he/she 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

if they assume that a considerable proportion of the other nodes follow the default tips selection strategy.

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.

It is worth noting that the attacker's task is very difficult also because of the network synchronization issues: he/she may not be aware about a good chunk of recently issued transactions[22]. Another effective way of fencing off such an attack would be that a sufficiently powerful entity publishes a large number of transactions at once (in one branch), thus rapidly changing the power's balance and making it difficult for the attacker to deal with this change. Observe also that, if the attacker manages to maintain the split, the recent transactions will only have around 50% of confirmation confidence (recall Section 1), and it will not grow; so, the "honest" nodes may decide to start approving only the pre-split transactions then (and, in particular, not approve neither of the two conflicting transactions).

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 some attack strategies, when the attacker tries to double-spend by "outpacing" the system.

2. The "large weight" attack means that, in order to double-spend, the attacker tries to give a very big weight to the double-spending transaction, so that alone it would outweigh the legitimate subtangle. This strategy would be indeed a menace to the network in case the allowed own weight is unbounded. As a

[22] so the "real" cumulative weights may be quite different from what he/she believes

solution, we may limit the own weight of a transaction from above, or even set it to constant.

3. In the situation when the maximal own weight of a transaction is m, the best attacker's strategy is to generate always transactions with the maximal own weight that reference the double-spending transaction. 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)).

4. 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.

5. 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.

5 Resistance to quantum computations

It is known that a (today still hypothetical) sufficiently large quantum computer can be very efficient for handling problems where only way to solve it is to guess answers repeatedly and check them. The process of finding a nonce in order to generate a Bitcoin block is a good example of such a problem. As of today, in average one must check around 2^{68} nonces to find a suitable hash that allows to generate a block. It is known (see e.g. [15]) that a quantum computer would need $\Theta(\sqrt{N})$ operations to solve a problem of the above sort that needs $\Theta(N)$ operations on a classical computer. Therefore, a quantum computer would be around $\sqrt{2^{68}} = 2^{34} \approx 17$ billion times more efficient in Bitcoin mining than a classical one. Also, it is worth noting that if blockchain does not increase its difficulty in response to increased hashing power, that would lead to increased rate of orphaned blocks.

Observe that, for the same reason, the "large weight" attack described above would also be much more efficient on a quantum computer. However, capping the weight from above (as suggested in Section 4) would effectively fence off a quantum computer attack as well, due to the following reason. In iota, the number of nonces

that one needs to check in order to find a suitable hash for issuing a transaction is not so huge, it is only around 3^8. The gain of efficiency for an "ideal" quantum computer would be therefore of order $3^4 = 81$, which is already quite acceptable (also, remember that $\Theta(\sqrt{N})$ could easily mean $10\sqrt{N}$ or so). Also, the algorithm is such that the time to find a nonce is not much larger than the time needed for other tasks necessary to issue a transaction, and the latter part is much more resistant against quantum computing.

Therefore, the above discussion suggests that the tangle provides a much better protection against an adversary with a quantum computer compared to the (Bitcoin) blockchain.

References

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

[2] bitcoinj. Working with micropayment channels. `https://bitcoinj.github.io/working-with-micropayments`

[3] 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)

[4] Moshe Babaioff, Shahar Dobzinski, Sigal Oren, Aviv Zohar (2012) On Bitcoin and red balloons. *Proc. 13th ACM Conf. Electronic Commerce*, 56–73.

[5] Richard Durrett (2004) Probability – Theory and Examples. *Duxbury advanced series.*

[6] Sergio Demian Lerner (2015) DagCoin: a cryptocurrency without blocks. `https://bitslog.wordpress.com/2015/09/11/dagcoin/`

[7] Yonatan Sompolinsky, Aviv Zohar (2013) Accelerating Bitcoin's Transaction Processing. Fast Money Grows on Trees, Not Chains. `https://eprint.iacr.org/2013/881.pdf`

[8] Yonatan Sompolinsky, Yoad Lewenberg, Aviv Zohar (2016) SPECTRE: Serialization of Proof-of-work Events: Confirming Transactions via Recursive Elections. `https://eprint.iacr.org/2016/1159.pdf`

[9] Yoad Lewenberg, Yonatan Sompolinsky, Aviv Zohar (2015) Inclusive Block Chain Protocols.
http://www.cs.huji.ac.il/~avivz/pubs/15/inclusive_btc.pdf

[10] Joseph Poon, Thaddeus Dryja (2016) The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments.
https://lightning.network/lightning-network-paper.pdf

[11] Sheldon M. Ross (2012) *Introduction to Probability Models.* 10th ed.

[12] David Vorick (2015) Getting rid of blocks. slides.com/davidvorick/braids

[13] Amir Dembo, Ofer Zeitouni (2010) *Large Deviations Techniques and Applications.* Springer.

[14] Sheldon M. Ross (2009) *A First Course in Probability.* 8th ed.

[15] Gilles Brassard, Peter Hyer, Alain Tapp (1998) Quantum cryptanalysis of hash and claw-free functions. *Lecture Notes in Computer Science* **1380**, 163–169.