The Tangle

Serguei Popov*

April 30, 2018. Version 1.4.3

Abstract

In this paper we analyze the mathematical foundations of IOTA, a cryptocurrency for the Internet-of-Things (IoT) industry. The main feature of this novel cryptocurrency is the *tangle*, a directed acyclic graph (DAG) for storing transactions. The tangle naturally succeeds the blockchain as its next evolutionary step, and offers features that are required to establish a machine-to-machine micropayment system.

An essential contribution of this paper is a family of Markov Chain Monte Carlo (MCMC) algorithms. These algorithms select attachment sites on the tangle for a transaction that has just arrived.

1 Introduction and description of the system

The rise and success of Bitcoin during the last six years proved that blockchain technology has real-world value. However, this technology also has a number of drawbacks that prevent it from being used as a generic platform for cryptocurrencies across the globe. One notable drawback is the concept of a transaction fee for transactions of any value. The importance of micropayments will increase in the rapidly developing IoT industry, and paying a fee that is *larger* than the amount of value being transferred is not logical. Furthermore, it is not easy to get rid of fees in the blockchain infrastructure since they serve as an incentive for the creators of blocks. This leads to another issue with existing cryptocurrency technology, namely the heterogeneous nature of the system. There are two distinct types of participants in the system, those who issue transactions, and those who approve transactions. The design of this system creates unavoidable discrimination of some participants, which in turn creates

*a.k.a. `mthcl`; author's contact information: `serguei.popov@iota.org`

conflicts that make all elements spend resources on conflict resolution. The aforementioned issues justify a search for solutions essentially different from blockchain technology, the basis for Bitcoin and many other cryptocurrencies.

In this paper we discuss an innovative approach that does not incorporate blockchain technology. This approach is currently being implemented as a cryptocurrency called *iota* [1], which was designed specifically for the IoT industry. The purpose of this paper is to focus on general features of the tangle, and to discuss problems that arise when one attempts to get rid of the blockchain and maintain a distributed ledger. The concrete implementation of the iota protocol is not discussed.

In general, a tangle-based cryptocurrency works in the following way. Instead of the global blockchain, there is a DAG that we call the *tangle*. The transactions issued by nodes constitute the site set of the tangle graph, which is the ledger for storing transactions. The edge set of the tangle 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 in Figure 1[2]. If there is not a 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 either directly or indirectly by all other transactions (Figure 2). The genesis is described in the following way. In the beginning of the tangle, there was an address with a balance that contained all of the tokens. The genesis transaction sent these tokens to several other "founder" addresses. Let us stress that all of the tokens were created in the genesis transaction. No tokens will be created in the future, and there will be no mining in the sense that miners receive monetary rewards "out of thin air".

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

The main idea of the tangle is the following: to issue a transaction, users must work to approve other transactions. Therefore, users who issue a transaction are contributing to the network's security. It is assumed that the nodes check if the approved transactions are not conflicting. If a node finds that a transaction is in conflict with the tangle history, the node will not approve the conflicting transaction in either a direct or indirect manner[3].

[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 have an entirely different set of rules.

[2]Time always increases from left to right in each figure.

[3]If a node issues a new transaction that approves conflicting transactions, then it risks that other nodes will not approve its new transaction, which will fall into oblivion.

As a transaction receives additional approvals, it is accepted by the system with a higher level of confidence. In other words, it will be difficult to make the system accept a double-spending transaction. It is important to observe that we do not *impose* any rules for choosing which transactions a node will approve. Instead, we argue that if a large number of nodes follow some "reference" rule, then for any fixed node it is better to stick to a rule of the same kind[4]. This seems to be a reasonable assumption, especially in the context of IoT, where nodes are specialized chips with pre-installed firmware.

In order to issue a transaction, a node does the following:

- The node chooses two other transactions to approve according to an algorithm. In general, these two transactions may coincide.
- The node checks if the two transactions are not conflicting, and does not approve conflicting transactions.
- For a node to issue a valid transaction, the node must solve a cryptographic puzzle similar to those in the Bitcoin blockchain. This is achieved by finding a nonce such that the hash of that nonce concatenated with some data from the approved transaction has a particular form. In the case of the Bitcoin protocol, the hash must have at least a predefined number of leading zeros.

It is important to observe that the iota network is asynchronous. In general, nodes do not necessarily see the same set of transactions. It should also be noted that the tangle may contain conflicting transactions. The nodes do not have to achieve consensus on which valid[5] transactions have the right to be in the ledger, meaning all of them can be in the tangle. However, in the case where there are conflicting transactions, the nodes need to decide which transactions will become orphaned[6]. The main rule that the nodes use for deciding between two conflicting transactions is the following: a node runs the tip selection algorithm[7] (cf. Section 4.1) many times, and sees which of the two transactions is more likely to be indirectly approved by the selected tip. For example, if a transaction was selected 97 times during 100 runs of the tip selection algorithm, 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? Every node calculates some statistics, one of which is

[4]We comment more on this at the end of Section 4.1

[5]Transactions that are issued according to the protocol.

[6]Orphaned transactions are not indirectly approved by incoming transactions anymore

[7]As mentioned above, there is a good reason to assume that other nodes would follow the same algorithm for tip selection.

how many new transactions are received from a neighbor. If one particular node is "too lazy", it will be dropped by its neighbors. Therefore, even if a node does not issue transactions, and hence has no direct incentive to share new transactions that approve its own transaction, it still has incentive to participate.

After introducing some notation in Section 2, we discuss algorithms for choosing the two transactions to approve, the rules for measuring the transaction's overall approval (Section 3, especially Section 3.1), and possible attack scenarios (Section 4). Also, in the unlikely event that the reader is scared by the formulas, they can jump directly to the "conclusions" at the end of each section.

It should be noted that the idea of using DAGs in the cryptocurrency space has been around for some time, see [3, 6, 7, 9, 12]. Specifically, [7] introduces the GHOST protocol, which proposes a modification of the Bitcoin protocol by making the main ledger a tree instead of a blockchain. It is shown that such a modification reduces confirmation times and improves the overall security of the network. In [9] the authors consider a DAG-based cryptocurrency model. Their model is different than our model for the following reasons: the sites of their DAG are blocks instead of individual transactions; the miners in their system compete for transaction fees; and new tokens may be created by block miners. Also, observe that a solution somewhat similar to ours was proposed in [6], although it does not discuss any particular tip approval strategies. After the first version of this paper was published, several other works that aim to create a DAG-based distributed ledger have appeared, e.g. [8]. We also reference another approach [2, 10] that aims to make Bitcoin micropayments possible by establishing peer-to-peer payment channels.

2 Weights and more

In this section we define the 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 the current implementation of iota, the weight may only assume values 3^n, where n is a positive integer that belongs to some nonempty interval of acceptable values[8]. In fact, it is irrelevant to know how the weight was obtained in practice. It is only important that every transaction has a positive integer, its weight, attached to it. In general, the idea is that a transaction with a larger weight is more "important" than a transaction with a smaller weight. To avoid spamming and other attack styles, it is assumed that no entity can generate an abundance of transactions with "acceptable" weights in a short period of time.

[8]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 a particular transaction plus the sum of own weights of all transactions that directly or indirectly approve this transaction. The algorithm for cumulative weight calculation is illustrated in Figure 1. The boxes represent transactions, the small number in the SE corner of each box denotes own weight, and the bold number denotes the cumulative weight. For example, transaction F is directly or indirectly approved by transactions A, B, C, E. The cumulative weight of F is $9 = 3 + 1 + 3 + 1 + 1$, which is the sum of the own weight of F and the own weights of A, B, C, E.

Let us define "tips" as unapproved transactions in the tangle graph. In the top tangle snapshot of Figure 1, the only tips are A and C. When the new transaction X arrives and approves A and C in the bottom tangle snapshot, X becomes the only tip. The cumulative weight of all other transactions increases by 3, the own weight of X.

We need to introduce two additional variables for the discussion of approval algorithms. First, for a transaction site on the tangle, we introduce its

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

For example, G has height 1 and depth 4 in Figure 2 because of the reverse path F, D, 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 the sum of own weights of all transactions approved by this transaction plus the own weight of the transaction itself. In Figure 2, the only tips are A and C. Transaction A directly or indirectly approves 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 order to understand the arguments presented in this paper, one may safely assume that all transactions have an own weight equal to 1. *From now on, we stick to this assumption.* Under this assumption, the cumulative weight of transaction X becomes 1 plus the number of transactions that directly or indirectly approve X, and the score becomes 1 plus the number of transactions that are directly or indirectly approved by X.

Let us note that, among those defined in this section, the cumulative weight is (by far!) the most important metric, although height, depth, and score will briefly enter some discussions as well.

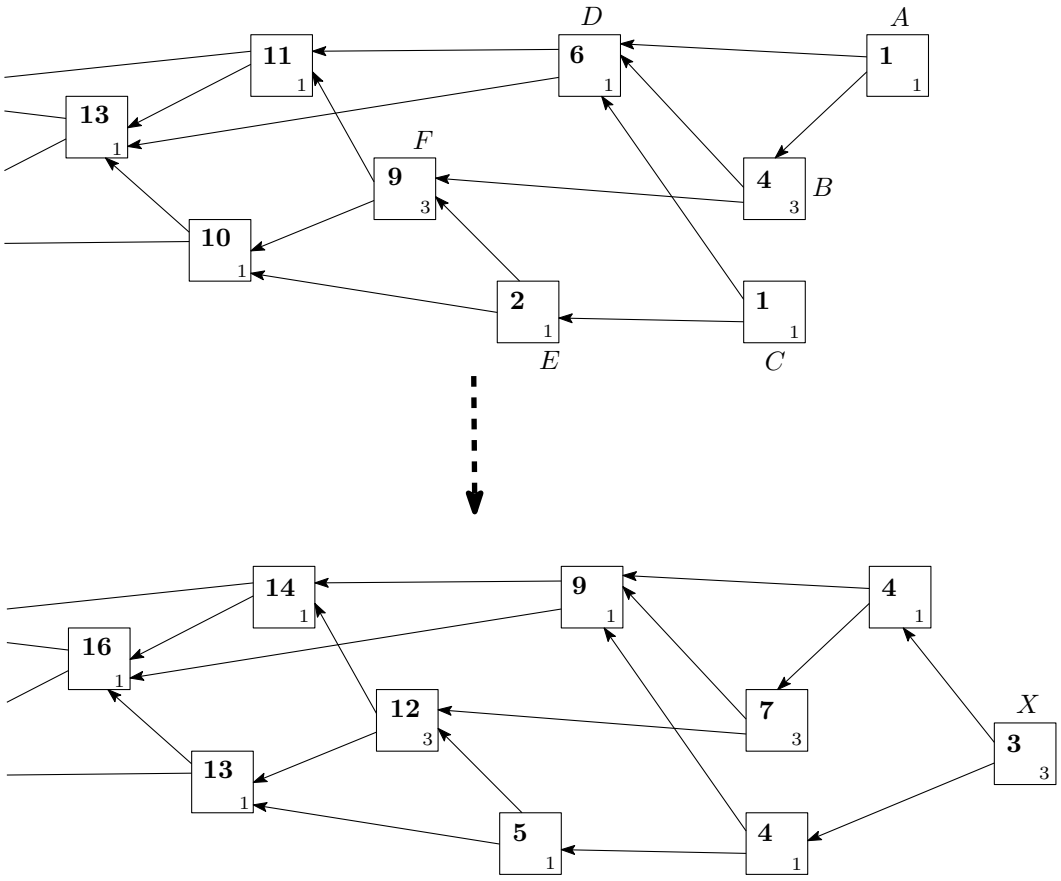

Figure 1: DAG with weight assignments before and after a newly issued transaction, X. The boxes represent transactions, the small number in the SE corner of each box denotes own weight, and the bold number denotes the cumulative weight.

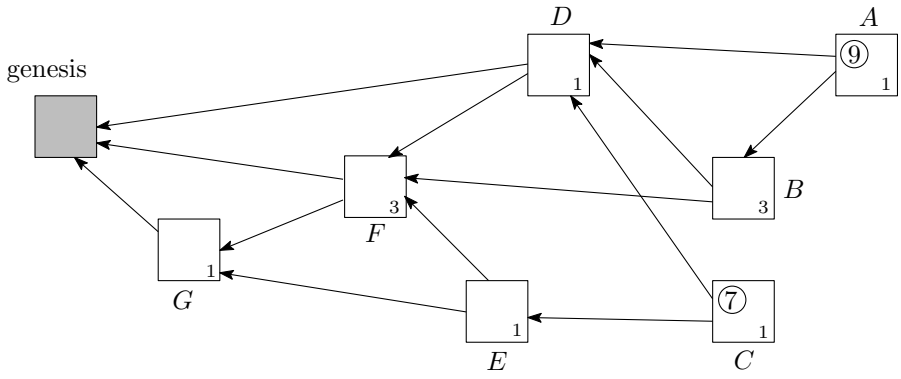

Figure 2: DAG with own weights assigned to each site, and scores calculated for sites A and C.

3 Stability of the system, and cutsets

Let $L(t)$ be the total number of tips in the system at time t. One expects that the stochastic process $L(t)$ remains *stable*[9]. More precisely, one expects the process to be *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. If $L(t)$ were to escape to infinity, many unapproved transactions would be left behind.

To analyze the stability properties of $L(t)$, we need to make some assumptions. One assumption is that transactions are issued by a 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 that this rate remains constant in time. Assume that all devices have approximately the same computing power, and let h be the average time a device needs to perform calculations that are required to issue a transaction. Then, let us *assume* that all nodes behave in the following way: to issue a transaction, a node 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 because it 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 this model since it is simple to analyze, and may provide insight into the system's behavior for more complicated tip selection strategies.

[9]Under an additional assumption that the process is time-homogeneous.

Next, we make a further simplifying assumption that any node, at the moment when it issues a transaction, observes not the actual state of the tangle, but the one exactly h time units ago. This means, in particular, that a transaction attached to the tangle at time t only becomes visible to the network at time $t+h$. We also assume that the number of tips remains roughly stationary in time, and is concentrated around a number $L_0 > 0$. In the following, we will calculate L_0 as a function of λ and h.

Observe that, at a given time t we have roughly λh "hidden tips" (which were attached in the time interval $[t-h, t)$ and so are not yet visible to the network); also, assume that typically there are r "revealed tips" (which were attached before time $t-h$ and remain tips at time t), so $L_0 = r + \lambda h$. By stationarity, we may then assume that at time t there are also around λh sites that were tips at time $t-h$, but are not tips anymore. Now, think about a new transaction that comes at this moment; then, a transaction it chooses to approve is a tip with probability $r/(r+\lambda h)$ (since there are around r tips known to the node that issued the transaction, and there are also around λh transactions which are not tips anymore, although that node thinks they are), so the mean number of chosen tips is $2r/(r+\lambda h)$. The key observation is now that, in the stationary regime, this mean number of chosen tips should be equal to 1, since, in average, a newcoming transaction should not change the number of tips. Solving the equation $2r/(r+\lambda h) = 1$ with respect to r, we obtain $r = \lambda h$, and so

$$L_0 = 2\lambda h. \tag{1}$$

We also note that, if the rule is that a new transaction references k transactions instead of 2, then a similar calculation gives

$$L_0^{(k)} = \frac{k\lambda h}{k-1}. \tag{2}$$

This is, of course, consistent with the fact that $L_0^{(k)}$ should tend to λh as $k \to \infty$ (basically, the only tips would be those still unknown to the network).

Also (we return to the case of two transactions to approve) the expected time for a transaction to be approved for the first time is approximately $h + L_0/2\lambda = 2h$. This is because, by our assumption, during the first h units of time a transaction cannot be approved, and after that the Poisson flow of approvals to it has rate approximately $2\lambda/L_0$. (Recall Proposition 5.3 of [11], which says that if we independently classify each event of a Poisson process according to a list of possible subtypes, then the processes of events of each subtype are independent Poisson processes.)

Observe that[10] at any fixed time t the set of transactions that were tips at some

[10] At least in the case where the nodes *try* to approve tips.

moment $s \in [t, t + h(L_0, N)]$ typically constitutes a *cutset*. 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 a new cutset in the tangle occasionally becomes small. One may then use the small cutsets as checkpoints for possible DAG pruning and other tasks.

It is important to observe that the above "purely random" approval strategy is not very good in practice because it does not encourage approving tips. A "lazy" user could always approve a fixed pair of very old transactions, therefore not contributing to the approval of more recent transactions, without being punished for such behavior[11]. Also, a malicious entity can artificially inflate the number of tips by issuing many transactions that approve a fixed pair of transactions. This would make it possible for future transactions to select these 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 that is biased towards the "better" tips. One example of such a strategy is presented in Section 4.1 below.

Before starting the discussion about the expected time for a transaction to receive its first approval, note that we can distinguish two regimes (Figure 3).

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

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

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 different extremes.

The situation in the low load regime is relatively simple. The first approval happens on an average timescale of order λ^{-1} since one of the first few incoming transactions will approve a given tip.

Let us now consider the high load regime, the case where L_0 is large. As mentioned above, one may assume that the Poisson flows of approvals to different tips are

[11] We remind the reader that we do not try to *enforce* any particular tip selection strategy. An attacker can choose tips in any way they find convenient.

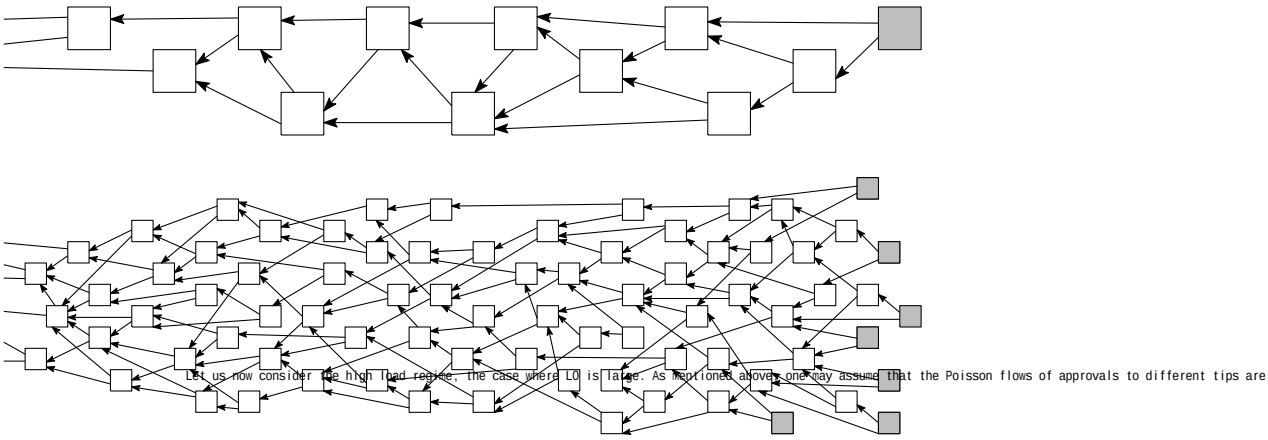

Figure 3: Low load (top) and high load (bottom) regimes of incoming transaction flow. White squares represent verified sites, while gray squares represent tips.

independent and have an approximate rate of $2\lambda/L_0$. Therefore, the expected time for a transaction to receive its first approval is around $L_0/(2\lambda) = h$ (recall (1)).

However, it is worth noting that for more elaborate approval strategies[12], it may not be a good idea to passively wait a long time until a transaction is approved by the others. This is due to the fact that "better" tips will keep appearing and will be preferred for approval. Rather, in the case when a transaction is waiting for approval over a time interval much larger than $L_0/2\lambda$, a good strategy would be to promote this latent transaction with an additional empty transaction[13]. In other words, a node can issue an empty transaction that approves its previous transaction together with one of the "better" tips to increase the probability that the empty transaction receives approval.

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 elaborate strategies[14] to defend against such attacks in that section. In the meantime, it is still

[12]That favor "better" quality tips in future implementations of iota.

[13]An empty transaction is a transaction that does not involve any token transfer, but still has to approve two other transactions. It should be noted that generating an empty transaction contributes to the network's security.

[14]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.

worth considering the simple tip selection strategy where an incoming transaction approves two random tips. This strategy is the easiest to analyze, and therefore may provide some insight into the qualitative and quantitative behavior of the tangle.

Conclusions:

1. We distinguish between two regimes, low load and high load (Figure 3).
2. There are only a few tips in the low load regime. A tip gets approved for the first time in $\Theta(\lambda^{-1})$ time units, 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 tip approval strategy employed by the new transaction.
4. If a transaction uses the strategy of approving two random tips, the typical number of tips is given by (1). It can be shown that this strategy is optimal with respect to the typical number of tips. However, it is not practical to adopt this strategy because it does not encourage approving tips.
5. More elaborate strategies are needed to handle attacks and other network issues. A family of such strategies is discussed in Section 4.1.
6. The typical time for a tip to be approved is $\Theta(h)$ in the high load regime, where h is the average computation/propagation time for a node. However, if the first approval does not occur in the above time interval, it is a good idea for the issuer and/or receiver to promote that transaction with an additional empty transaction.

3.1 How fast does the cumulative weight typically grow?

Assume that the network is in the low load regime. After a transaction gets approved several times, its cumulative weight will grow with speed λ because all new transactions will indirectly reference this transaction[15].

In the case where the network is in the high load regime, an old transaction with a large cumulative weight will experience weight growth with speed λ because essentially all new transactions will indirectly reference it. Moreover, when the transaction

[15]Recall that we assumed that the own weights of all transactions are equal to 1, so the cumulative weight is just the number of transactions that directly or indirectly reference a transaction plus 1.

is first added to the tangle it may have to wait for some time to be approved. In this time interval, the transaction's cumulative weight behaves in a random fashion. To characterize the speed with which the cumulative weight grows after the transaction receives several approvals, let us define $H(t)$ as the expected cumulative weight at time t (for simplicity, we start counting time at the moment when our transaction was revealed to the network, i.e., h time units after it was created) and $K(t)$ as the expected number of tips that approve the transaction at time t. Let us also abbreviate $h := h(L_0, N)$. We make a simplifying assumption that the number of tips remains roughly constant at a value of L_0 over time. We work with the "approve two random tips" strategy in this section. It is expected that the qualitative behavior will be roughly the same for other reasonable strategies.

Recall that a transaction entering the network at time t typically chooses two tips to approve based on the state of the system at time $t-h$ because the node must do some calculations and verifications before actually issuing the transaction. It is not difficult to see that (assuming, though, that $K(\cdot)$ is the *actual* number of tips, not just expected number) the probability of the transaction approving at least one of "our" tips in the tangle is $1-\left(1-\frac{K(t-h)}{L_0}\right)^2 = \frac{K(t-h)}{L_0}\left(2-\frac{K(t-h)}{L_0}\right)$[16]. Analogous to Example 6.4 of [11], we can write for small $\delta > 0$

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

and thus deduce the following differential equation

$$\frac{dH(t)}{dt} = \lambda\frac{K(t-h)}{L_0}\left(2-\frac{K(t-h)}{L_0}\right). \tag{3}$$

In order to be able to use (3), we need to first calculate $K(t)$. This is not a trivial task since a tip at time $t-h$ may not be a tip at time t, and the overall number of tips approving the original transaction increases by 1 in the case where an incoming transaction approves such a tip. The crucial observation is that the probability that a tip at time $t-h$ remains a tip at time t is approximately $1/2$. (To verify this, recall the discussion from Section 3: the typical number of tips is $2\lambda h$, and during the interval of length h new λh tips will substitute a half of old ones.) Therefore, at time t approximately one half $K(t-h)$ tips remain in the unconfirmed tip state, while the other half will have received at least one approval. Let A denote the set of $K(t-h)/2$ tips at time $t-h$ that are still tips at time t, and let B denote the

[16]The expression on the left-hand side is 1 minus the probability that the two approved tips are not ours.

remaining set of $K(t-h)/2$ tips that were already approved by time t. Let p_1 be the probability that a new transaction approves at least 1 transaction from B and does not approve any transactions from A. Furthermore, let p_2 be the probability that both approved transactions belong to A. In other words, p_1 and p_2 are 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 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 $A \cup B$. Analogous to (3), the differential equation for $K(t)$ is:

$$\frac{dK(t)}{dt} = (p_1 - p_2)\lambda = \lambda \frac{K(t-h)}{L_0}\Big(1 - \frac{K(t-h)}{L_0}\Big). \tag{4}$$

It is difficult to solve (4) 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 $(1-\varepsilon)L_0$. Now, when $K(t)$ is small with respect to L_0, we can drop the last factor in the right-hand side of (4)[17]. We obtain a simplified version of (4) by recalling that $\frac{\lambda h}{L_0} = \frac{1}{2}$:

$$\frac{dK(t)}{dt} \approx \frac{1}{2h} K(t-h), \tag{5}$$

with boundary condition $K(0) = 1$. We look for a solution of the form $K(t) = \exp(c\frac{t}{h})$; after substituting this into (5), we obtain

$$\frac{c}{h}\exp\Big(c\frac{t}{h}\Big) \approx \frac{1}{2h}\exp\Big(c\frac{t}{h} - c\Big),$$

therefore

$$K(t) = \exp\Big(W\big(\tfrac{1}{2}\big)\frac{t}{h}\Big) \approx \exp\Big(0.352\frac{t}{h}\Big) \tag{6}$$

is an approximate solution, where $W(\cdot)$ is the so-called Lambert W-function.[18] Taking the logarithm of both sides in (6), we find that the time when $K(t)$ reaches εL_0 is

[17] It would be a constant close to 1, so the right-hand side would be equivalent to $\lambda\frac{K(t-h)}{L_0}$.

[18] Also known as the omega function or product logarithm; for $x \in [0, +\infty)$ it is characterized by the relation $x = W(x)\exp(W(x))$.

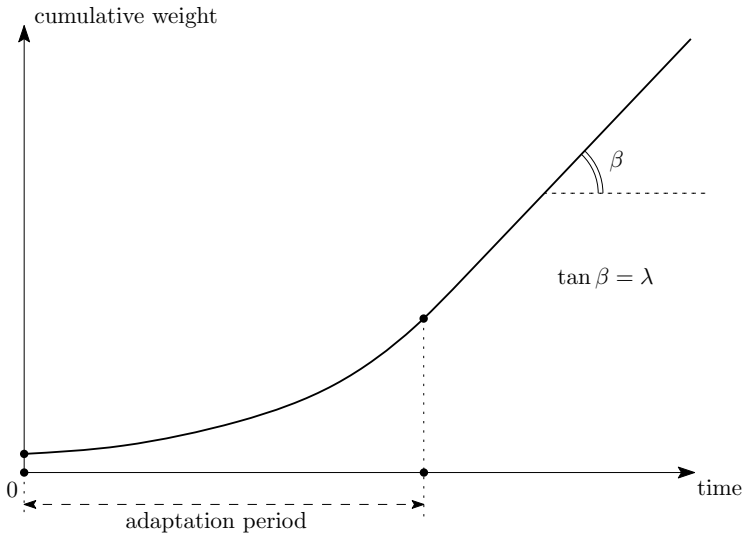

Figure 4: Plot of cumulative weight vs. time for the high load regime.

roughly

$$t_0 \approx \frac{h}{W\left(\frac{1}{2}\right)} \times \left(\ln L_0 - \ln \varepsilon^{-1}\right) \lesssim 2.84 \cdot h \ln L_0. \tag{7}$$

Returning to (3) and dropping the last term on the right-hand side, we obtain that during the "adaptation period" (i.e., $t \leq t_0$ with t_0 as in (7)), it holds that

$$\begin{aligned} \frac{dH(t)}{dt} &\approx \frac{2\lambda}{L_0} K(t-h) \\ &\approx \frac{1}{h \exp\left(W(\frac{1}{2})\right)} \exp\left(W\left(\tfrac{1}{2}\right)\frac{t}{h}\right) \\ &= \frac{2W\left(\frac{1}{2}\right)}{h} \exp\left(W\left(\tfrac{1}{2}\right)\frac{t}{h}\right) \end{aligned}$$

and therefore

$$H(t) \approx 2\exp\left(W\left(\tfrac{1}{2}\right)\frac{t}{h}\right) \approx 2\exp\left(0.352\frac{t}{h}\right). \tag{8}$$

Let us also remind the reader that after the adaptation period, the cumulative weight $H(t)$ grows linearly with speed λ. We stress that the "exponential growth" in (8) does not mean that the cumulative weight grows "very quickly" during the adaptation period. Rather, the behavior is as depicted in Figure 4.

Conclusions:

1. After a transaction gets approved multiple times in the low load regime, its cumulative weight will grow with speed λw, where w is the mean weight of a generic transaction.

2. In the high load regime, there are two distinct growth phases. First, a transaction's cumulative weight $H(t)$ grows with increasing speed during the *adaptation period* according to (8). After the adaptation period is over, the cumulative weight grows with speed λw (Figure 4). In fact, for *any* reasonable strategy the cumulative weight will grow with this speed after the end of the adaptation period because all incoming transactions will indirectly approve the transaction of interest.

3. One can think of the adaptation period of a transaction as the time until most of the current tips indirectly approve that transaction. The typical length of the adaptation period is given by (7).

4 Possible attack scenarios

We start by discussing an attack scenario where the attacker tries to "outpace" the network alone:

1. An attacker sends a payment to a merchant and receives the goods after the merchant decides the transaction has a sufficiently large cumulative weight.

2. The attacker issues a double-spending transaction.

3. The attacker uses their computing power to issue many small transactions that approve the double-spending transaction, but do not approve the original transaction that they sent to the merchant either directly or indirectly.

4. It is possible for the attacker to have a plethora of Sybil identities which are not required to approve tips.

5. An alternative method to item 3 would be for the attacker to issue a big double-spending transaction using all of their computing power. This transaction would have a very large own weight[19], and would approve transactions prior to the legitimate transaction used to pay the merchant.

[19]Here we assume that the own weight of a transaction may vary. It will become clear in the discussion below why it is a good idea to let the own weight vary.

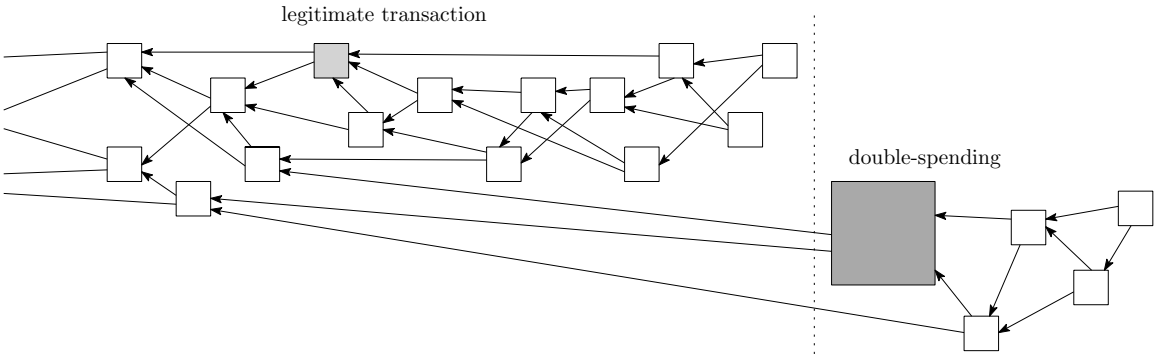

Figure 5: The "large weight" attack

6. The attacker hopes that their dishonest subtangle outpaces the honest subtangle. If this happens, the main tangle continues growing from the double-spending transaction, and the legitimate branch with the original payment to the merchant is orphaned (Figure 5).

In fact, it can be shown that the strategy of one large double-spending transaction increases the attacker's chances of being successful. In the "ideal" situation of this mathematical model, this attack *always* succeeds.

Let $W^{(n)}$ be the time needed to obtain a nonce that gives the double-spending transaction a weight of at least 3^n. One may assume that $W^{(n)}$ is an exponentially distributed random variable with parameter[20] $\mu 3^{-n}$, where μ represents the computing power of the attacker.

Assume that the merchant accepts the legitimate transaction when its cumulative weight becomes at least w_0, which happens t_0 time units after the original transaction. It is reasonable to expect that the cumulative weight grows with linear speed λw, where λ is the overall arrival rate of transactions issued on the network by honest nodes, and w is the mean weight of a generic transaction. The typical total weight of the legitimate branch at that time is $w_1 = \lambda w t_0$.

Let $\lceil x \rceil$ be the smallest integer greater than or equal to x, define $n_0 = \left\lceil \frac{\ln w_1}{\ln 3} \right\rceil$, so that $3^{n_0} \geq {w_1}$[21]. If the attacker managed to obtain a nonce that gives the double-spending transaction a weight of at least 3^{n_0} during the time interval of length t_0, then the attack succeeds. The probability of this event is

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

[20] With expectation $\mu^{-1} 3^n$.

[21] In fact, $3^{n_0} \approx w_1$ if w_1 is large.

This approximation is true in the case where $\frac{t_0\mu}{w_1}$ is small, which is a reasonable assumption. If this "immediate" attack does 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 they find it, the total weight of the legitimate branch is smaller than 3^n. The probability of this event occurring 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 typically be a small number, at each "level" n the attack succeeds with a constant probability. Therefore, it will a.s. succeed. The typical time until it succeeds is roughly $3^{\frac{\lambda w}{\mu}}$. Although this quantity may be very large, the probability that the "first"[22] attack succeeds is not negligible. Therefore, we need countermeasures. One such countermeasure would be limiting the own weight from above, or even setting it to a constant value. As mentioned in Section 3, the latter may not be the best solution because it does not offer enough protection from spam.

Now, let us discuss the situation where the maximum own weight is capped at a value of 1, 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, and that the adaptation period for that transaction is over. In this situation, the transaction's cumulative weight increases linearly with speed λ. Now, imagine that the attacker wants to double-spend on this transaction. To do so, the attacker secretly prepares the double-spending transaction, and starts generating *nonsense* transactions that approve the double-spending transaction at the time[23] when the *original* transaction was issued to the merchant. If the attacker's subtangle outpaces the legitimate subtangle at some moment after the merchant decides to accept the legitimate transaction, then the double-spending attack would be successful. If that does not happen, then the double-spending transaction would not be approved by others because the legitimate transaction would acquire more cumulative weight and essentially all new tips would indirectly approve it. The double-spending transaction would be orphaned in this scenario.

As before, let μ stand for the computing power of the attacker. 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[24], and define $V_k = \mu G_k$, $k \geq 1$. It follows that $V_1, V_2, V_3, \ldots$ are i.i.d. exponential random variables with parameter 1.

[22] During the time t_0.

[23] Or even before; we discuss this case later.

[24] With expected value $1/\mu$.

Suppose that at time t_0 the merchant decides to accept the transaction with cumulative weight w_0. Let us estimate the probability that the attacker successfully double-spends. Let $M(\theta) = (1-\theta)^{-1}$ be the moment generating function of the exponential distribution with parameter 1 (Section 7.7 of [14]). It is known[25] that for $\alpha \in (0,1)$ it holds that

$$\mathbb{P}\Big[\sum_{k=1}^{n} V_k \leq \alpha n\Big] \approx \exp\big(-n\varphi(\alpha)\big), \tag{9}$$

where $\varphi(\alpha) = -\ln\alpha + \alpha - 1$ is the Legendre transform of $\ln M(\theta)$. 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 that $\frac{\mu t_0}{w_0} < 1$, otherwise the probability that the attacker's subtangle eventually outpaces the legitimate subtangle 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 transactions with maximum own weight m during time t_0. Therefore, using (9), we find the probability that the double-spending transaction has more cumulative weight at time t_0 is roughly

$$\begin{aligned}\mathbb{P}\Big[\sum_{k=1}^{w_0/m} G_k < t_0\Big] &= \mathbb{P}\Big[\sum_{k=1}^{w_0} V_k < \mu t_0\Big] \\ &= \mathbb{P}\Big[\sum_{k=1}^{w_0} V_k < w_0 \times \frac{\mu t_0}{w_0}\Big] \\ &\approx \exp\big(-w_0\varphi\big(\tfrac{\mu t_0}{w_0}\big)\big). \end{aligned} \tag{10}$$

For the above probability to be small, $\frac{w_0}{m}$ needs to be large and $\varphi\big(\frac{\mu t_0}{w_0}\big)$ cannot be very small.

Note that, at time $t \geq t_0$, the cumulative weight of the legitimate transaction is roughly $w_0 + \lambda(t-t_0)$ because we assumed that the adaptation period is over, so the cumulative weight grows with speed λ. Analogous to (10), one finds the probability that the double-spending transaction has more cumulative weight at time $t \geq t_0$ is roughly

$$\exp\big(-(w_0+\lambda(t-t_0))\varphi\big(\tfrac{\mu t}{w_0+\lambda(t-t_0)}\big)\big). \tag{11}$$

Then, it must be true that we have $\frac{\mu t_0}{w_0} \geq \frac{\mu}{\lambda}$ since the cumulative weight grows with speed less than λ during the adaptation period. It can be shown that the probability

[25] This is a consequence of the so-called Large Deviation Principle. See the general book [13], and 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.

of achieving a successful double spend is of order

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

For example, let $\mu = 2$, $\lambda = 3$ so that the attacker's power is only a bit less than that of the rest of the network. Assume that the transaction has a cumulative weight of 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 (12) then gives the upper bound approximately 0.29. If one assumes that $\mu = 1$ and keeps all 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 (12) gives approximately 0.00001135, quite a drastic change.

From the above discussion it is important to recognize that the inequality $\lambda > \mu$ should be true for the system to be secure. In other words, the input flow of "honest" transactions should be large compared to the attacker's computational power. Otherwise, the estimate (12) would be useless. This indicates the need for additional security measures, such as checkpoints, during the early days of a tangle-based system.

When choosing a strategy for deciding which one of two conflicting transactions is valid, one has to be careful when using cumulative weight as a decision metric. This is due to the fact that cumulative weight can be subject to an attack similar to the one described in Section 4.1, namely the attacker may prepare a double-spending transaction well in advance, build a secret subtangle referencing it, and then broadcast that subtangle after the merchant accepts the legitimate transaction. 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 of the two transactions 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 subtangle that occasionally references the main tangle to gain a higher score. Note that the score of honest 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. Since network latency is not an issue for an attacker who builds a subtangle alone[26], they might be able to give more height to the parasite tips if they use a computer that is sufficiently strong. Moreover, the attacker can artificially increase their tip count at the moment of the attack by broadcasting many new transactions

[26]This is due to the fact that an attacker can always approve their own transactions without relying on any information from the rest of the network.

that approve transactions that they issued earlier on the parasite chain (Figure 6). This will give the attacker an advantage in the case where the honest nodes use some selection strategy that involves a simple choice between available tips.

To defend against this attack style, we are going to use the fact that the main tangle is supposed to have more active hashing power than the attacker. Therefore, the main tangle is able to produce larger increases in cumulative weight for more transactions than the attacker. The idea is to use a MCMC algorithm to select the two tips to reference.

Let $\mathcal{H}_x$ be the current cumulative weight of a site. Recall that we assumed all own weights are equal to 1. Therefore, 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[27] way. The tips "chosen" by the walks are then the candidates for approval. The algorithm is described in the following way:

1. Consider all sites on the interval $[W, 2W]$, where W is reasonably large[28].

2. Independently place N particles on sites in that interval[29].

3. Let these particles perform independent discrete-time random walks "towards the tips", meaning that a transition from x to y is possible if and only if y approves x

4. The two random walkers that reach the tip set first will sit on the two tips that will be approved. However, it may be wise to modify this rule in the following way: first discard those random walkers that reached the tips *too fast* because they may have ended on one of the "lazy tips".

5. The transition probabilities of the walkers are defined in the following way: if y approves x ($y \rightsquigarrow x$), then the transition probability P_{xy} is proportional to

[27] There is not a "canonical" source of randomness. The nodes just use their own (pseudo)random number generators to simulate the random walks.

[28] The idea is to place the particle "deep" into the tangle so that it will not arrive at a tip straight away. However, the particle should not be placed "too deep" because it needs to find a tip in a reasonable time. Also, the interval $[W, 2W]$ is arbitrary. One could chose $[W, 5W]$, etc. There are also other ways to select the walkers' starting points. For example, a node can simply take a random transaction received between t_0 and $2t_0$ time units in the past, where t_0 is some fixed time point.

[29] This choice is largely arbitrary. We use several particles instead of just two for additional security. The idea is that if a particle were to accidentally jump to the attacker's chain, which is supposed to be long, then it would spend a lot of time there and other tips will be chosen first.

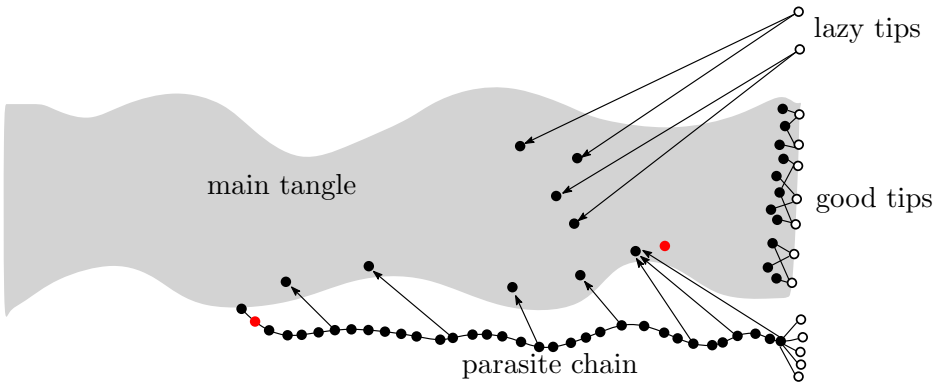

Figure 6: Visual representation of the tip selection algorithm for honest tips, as well as the parasite chain. The two red circles indicate an attempted double-spend by an attacker.

$\exp\big(-\alpha(\mathcal{H}_x-\mathcal{H}_y)\big)$, that is

$$P_{xy}=\exp\big(-\alpha(\mathcal{H}_x-\mathcal{H}_y)\big)\Big(\sum_{z:z\rightsquigarrow x}\exp\big(-\alpha(\mathcal{H}_x-\mathcal{H}_z)\big)\Big)^{-1}, \tag{13}$$

where $\alpha>0$ is a parameter to be chosen[30].

Note that this algorithm is "local", meaning one does not need to traverse the tangle back to the genesis to perform relevant calculations. In particular, observe that one does not need to calculate the cumulative weights for the whole tangle. At most one needs to calculate the cumulative weights for the sites that indirectly approve the starting point of the walker.

To check that the algorithm works as intended, first consider the "lazy tips". These tips intentionally approve some old transactions to avoid doing verification work (Figure 6). Even if the particle is on a site approved by a lazy tip, it is not probable that the lazy tip would be selected because the difference between cumulative weights would be very large and P_{xy} would be small.

Next, consider this alternate attack style: the attacker secretly builds a chain containing a transaction that empties their account balance to another account under their control, indicated as the leftmost red circle in Figure 6. Then, the attacker issues a transaction on the main tangle, represented by the rightmost red circle, and waits for the merchant to accept it. The parasite chain occasionally references the main

[30] One can start with $\alpha=1$.

tangle. However, the cumulative weight is not very large in the parasite chain. It should be noted that the parasite chain cannot reference the main tangle after the merchant's transaction. Furthermore, the attacker might try to artificially inflate the number of tips in their parasite chain at the moment of the attack (Figure 6). The attacker's idea is to make the nodes issuing new transactions reference the parasite chain so that the honest branch of the tangle will be orphaned.

It is easy to see why the MCMC selection algorithm will not select one of the attacker's tips with high probability. The reasoning is identical to the lazy tip scenario: the sites on the parasite chain will have a cumulative weight that is much smaller than the sites that they reference on the main tangle. Therefore, it is not probable that the random walker will ever jump to the parasite chain unless it begins there, and this event is not very probable either because the main tangle contains more sites.

As an additional protecting measure, we can first run a random walk with a large α (so that it is in fact "almost deterministic") to choose a "model tip"; then, use random walks with small α for actual tip selection, but verify if the (indirectly) referenced transactions are consistent with the model tip.

Observe also that, for a random walk that *always* moves towards the tips it is very simple and rapid to calculate the exit probability distribution using a straightforward recursion; this is something that we *do not* want the nodes to do. However, it is possible to modify our approach in the following way: on each step, the random walk may backtrack (i.e., go 1 step away from the tips) with probability (say) $\frac{1}{3}$ (and divide the remaining $\frac{2}{3}$ as before). The walk will reach the tips very quickly anyway (because it has a drift towards the tips), but it will not be so easy to calculate the exit measure.

Let us comment on why the nodes would follow this algorithm. Recall from Section 1 that 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 a transaction is issued. *It may be a good idea to intentionally move this snapshot to a time point further in the past*[31] in the reference algorithm for the reasons that we explain in the sequel. Imagine a "selfish" node that just wants to maximize the chances of their transaction being approved quickly. The MCMC algorithm of this section, which is adopted by a considerable proportion of nodes, defines a probability distribution on the set of tips. It is clear that

[31] First the random walker finds a former tip with respect to that snapshot, and then it continues to walk towards the "actual" tips on the current tangle.

a natural first choice for a selfish node would be to choose the tips where the maximum of that distribution is attained. However, if many other nodes also behave in a selfish way and use the same strategy, which is a reasonable assumption, then they all will lose. *Many* new transactions will approve the same two tips at roughly the same time, therefore generating too much competition between them for subsequent approval. It should also be clear that nodes will not immediately "feel" the cumulative weight increase caused by this mass approval of the same two tips since the nodes are using a past snapshot. For this reason, even a selfish node would have to use some random tip approval algorithm[32] with a probability distribution for tip selection that is close to the default probability distribution produced by the reference tip selection algorithm. We do not claim that this "aggregated" probability distribution would be equal to the default probability distribution in the presence of selfish nodes. However, the above argument shows that it should be close to it. This means that the probability of many nodes attempting to verify the same "bad" tips would remain small. In any case, there is not a large incentive for the nodes to be selfish because possible gains only amount to a slight decrease in confirmation time. This is inherently different from other decentralized constructs, such as Bitcoin. The important fact is that nodes do not have reasons to abandon the MCMC tip selection algorithm.

We would like to mention that the definition of transition probabilities, as given in (13), has not been set in stone. Instead of the exponent, one can use a different function that decreases rapidly, such $f(s) = s^{-3}$. There is also freedom for choosing W and N as well. At this point in time, it is unclear if there are any theoretical arguments that show exactly in which way these parameters should be chosen. In sum, we feel that the main contribution of this section is the idea of using MCMC for tip selection.

4.2 Splitting attack

Aviv Zohar suggested the following attack scheme against the proposed MCMC algorithm. In the high-load regime, an attacker can try to split the tangle into two branches and maintain the balance between them. This would allow both branches to continue to grow. The attacker must place at least two conflicting transactions

[32]as noticed before, for a backtracking walk 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 MCMC 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 if they assume that a considerable proportion of the other nodes follow the default tips selection strategy.

at the beginning of the split to prevent an honest node from effectively joining the branches by referencing them both simultaneously. Then, the attacker hopes that roughly half of the network would contribute to each branch so that they would be able to "compensate" for random fluctuations, even with a relatively small amount of personal computing power. If this technique works, the attacker would be able to spend the same funds on the two branches.

To defend against such an attack, one needs to use a "sharp-threshold" rule that makes it too hard to maintain the balance between the two branches. An example of such a rule is selecting the longest chain on the Bitcoin network. Let us translate this concept to the tangle when it is undergoing a splitting attack. Assume that the first branch has total weight 537, and the second branch has total weight 528. If an honest node selects the first branch with probability very close to 1/2, then the attacker would probably be able to maintain the balance between the branches. However, if an honest node selects the first branch with probability much larger than 1/2, then the attacker would probably be unable to maintain the balance. The inability to maintain balance between the two branches in the latter case is due to the fact that after an inevitable random fluctuation, the network will quickly choose one of the branches and abandon the other. In order to make the MCMC algorithm behave this way, one has to choose a very rapidly decaying function f, and initiate the random walk at a node with large depth so that it is highly probable that the walk starts before the branch bifurcation. In this case, the random walk would choose the "heavier" branch with high probability, even if the difference in cumulative weight between the competing branches is small.

It is worth noting that the attacker's task is very difficult because of network synchronization issues: they may not be aware of a large number of recently issued transactions[33]. Another effective method for defending against a splitting attack would be for a sufficiently powerful entity to instantaneously publish a large number of transactions on one branch, thus rapidly changing the power balance and making it difficult for the attacker to deal with this change. If the attacker manages to maintain the split, the most recent transactions will only have around 50% confirmation confidence (Section 1), and the branches will not grow. In this scenario, the "honest" nodes may decide to start selectively giving their approval to the transactions that occurred before the bifurcation, bypassing the opportunity to approve the conflicting transactions on the split branches.

One may consider other versions of the tip selection algorithm. For example, if a node sees two big subtangles, then it chooses the one with a larger sum of own

[33]The "real" cumulative weights may be quite different from what they believe.

weights before performing the MCMC tip selection algorithm outlined above.

The following idea may be worth considering for future implementations. One could make the transition probabilities defined in (13) depend on both $\mathcal{H}_x - \mathcal{H}_y$ and $\mathcal{H}_x$ in such a way that the next step of the Markov chain is almost deterministic when the walker is deep in the tangle, yet becomes more random when the walker is close to tips. This will help avoid entering the weaker branch while assuring sufficient randomness when choosing the two tips to approve.

Conclusions:

1. We considered attack strategies for when an 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 large weight to the double-spending transaction so that it would outweigh the legitimate subtangle. This strategy would be a menace to the network in the case where the allowed own weight is unbounded. As a solution, we may limit the own weight of a transaction from above, or set it to a constant value.

3. In the situation where the maximal own weight of a transaction is m, the best attack strategy is to generate transactions with own weight m 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 a larger cumulative weight can be estimated using the formula (12) (see also examples below (12)).

4. The attack method of building a "parasite chain" makes approval strategies based on height or score obsolete since the attacker's sites will have higher values for these metrics when compared to the legitimate tangle. On the other hand, the MCMC tip selection algorithm described in Section 4.1 seems to provide protection against this kind of attack.

5. The MCMC tip selection algorithm also offers protection against the lazy nodes as a bonus.

5 Resistance to quantum computations

It is known that a sufficiently large quantum computer[34] could be very efficient for handling problems that rely on trial and error to find a solution. The process of finding a nonce in order to generate a Bitcoin block is a good example of such a problem. As of today, one must check an average of 2^{68} nonces to find a suitable hash that allows a new block to be generated. It is known (see e.g. [15]) that a quantum computer would need $\Theta(\sqrt{N})$ operations to solve a problem that is analogous to the Bitcoin puzzle stated above. This same problem would need $\Theta(N)$ operations on a classical computer. Therefore, a quantum computer would be around $\sqrt{2^{68}} = 2^{34} \approx 17$ billion times more efficient at mining the Bitcoin blockchain than a classical computer. Also, it is worth noting that if a blockchain does not increase its difficulty in response to increased hashing power, there would be an increased rate of orphaned blocks.

For the same reason, a "large weight" attack would also be much more efficient on a quantum computer. However, capping the weight from above, as suggested in Section 4, would effectively prevent a quantum computer attack as well. This is evident in iota because the number of nonces that one needs to check in order to find a suitable hash for issuing a transaction is not unreasonably large. On average, it is around 3^8. The gain of efficiency for an "ideal" quantum computer would therefore be of order $3^4 = 81$, which is already quite acceptable[35]. More importantly, the algorithm used in the iota implementation is structured such that the time to find a nonce is not much larger than the time needed for other tasks that are necessary to issue a transaction. The latter part is much more resistant against quantum computing, and therefore gives the tangle much more protection against an adversary with a quantum computer when compared to the (Bitcoin) blockchain.

Acknowledgements

The author thanks Bartosz Kusmierz, Cyril Grünspan, Olivia Saa, Razvan Savu, Samuel Reid, Toru Kazama, Rafael Kallis, and Rodrigo Bueno who pointed out several errors in earlier drafts, and James Brogan for his contributions towards making this paper more readable.

[34] Still a hypothetical construct as of today.

[35] Note that $\Theta(\sqrt{N})$ could easily mean $10\sqrt{N}$.

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.