WHITE PAPER

the new era of blockchain 4.0

iDAG

iDAG

TABLE OF CONTENT

ABSTRACT

The main feature of this cryptocurrency is the hive, a directed acyclic graph (DAG) for storing transactions. The hive 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 hive for a transaction that has just arrived.

The edge set of the hive is obtained in the following way: when a new transaction arrives, it must approve or try to approve (we will discuss below) two previous transactions. these approvals are represented by directed edges, as shown in Figure 1. 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. The genesis is described in the following way. In the beginning of the hive, 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 hive graph. The network is composed of nodes; that is, nodes are entities that issue and validate transactions.

The main idea of the hive 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 hive history, the node will not approve the conflicting transaction in either a direct or indirect manner.

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.

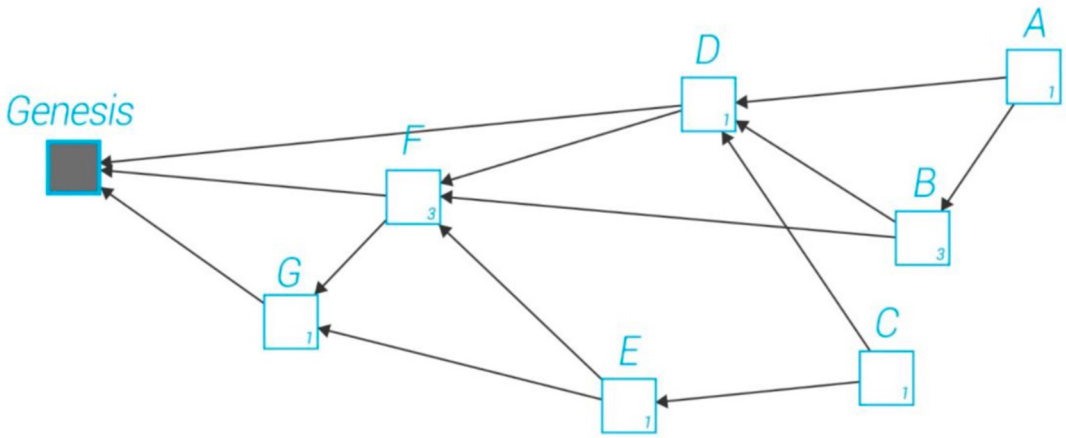
It is important to observe that we do not impose any rules for choosing which transactions a node will approve. In order to issue a transaction, a node does the following:

- 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 IDAG network is asynchronous. In general, nodes do not necessarily see the same set of transactions. It should also be noted that the hive may contain conflicting transactions. The nodes do not have to achieve consensus on which valid transactions have the right to be in the ledger, meaning all of them can be in the hive. However, in the case where there are conflicting transactions, the nodes need to decide which transactions will become orphaned.

The main rule that the nodes use for deciding between two conflicting transactions is the following: a node runs the tip selection algorithm many times, and sees which of the two transactions is more likely to be indirectly approved by the selected tip.

Let us also comment on the following question: what motivates the nodes to propagate transactions? 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 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.



DAG

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 IDAG, the weight may only assume values 3n, where n is a positive integer that belongs to some nonempty interval of acceptable values. In fact, it is irrelevant to know how the weight was obtained in practice. In general, the idea is that a transaction with a larger weight is more “important” than a transaction with a smaller weight.

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.

Let us define “tips” as unapproved transactions in the hive graph. We need to introduce two additional variables for the discussion of approval algorithms. First, for a transaction site on the hive, we introduce its height - the length of the longest oriented path to the genesis and depth - the length of the longest reverse-oriented path to some tip.

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 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.

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. More precisely, one expects the process to be positive recurrent. In particular, positive recurrence implies that the limit of $P[L(t) = k]$ as $t \to \infty$ should exist and be positive for all $k \geqslant 1$. 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. 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. 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.

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 hive, but the one exactly h time units ago. This means, in particular, that a transaction attached to the hive 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 $L0 > 0$. In the following, we will calculate L0 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 $t - h$), so $L0 = \lambda h + r$.

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 new coming 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 $L0 = 2\lambda h$(1).

We also note that, if the rule is that a new transaction references k transactions instead of 2, then a similar calculation gives $L0(k) = k\lambda h$(2).

This is, of course, consistent with the fact that L0(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 + L0/2 λ = 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 2λ/L0.

Observe that at any fixed time t the set of transactions that were tips at some moment ∗s ∈ [t; t + h(L0, 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 hive 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.

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.

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

- 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 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 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 $L_0/(2\lambda) = 1.45h(1)$.

However, it is worth noting that for more elaborate approval strategies, 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.

POOLS

To speed up the process of approving transactions, the system has group of nodes that are engaged in joint validation. It works like this: when a node is connected to the system, the network tells it which group it belongs to. After that, the time synchronization of all pool members takes place. At certain intervals, the network selects the most important transaction and report it to the pool, after that, each member of the pool begins confirmation. Let us explain, let n be the number of pool members, d the average number of iterations required for finding the nonce, i is the index of the pool member. Then, r = [(d / n) i; (d / n) i + d) the range of values that the i-th member of the pool needs to go over.

POW

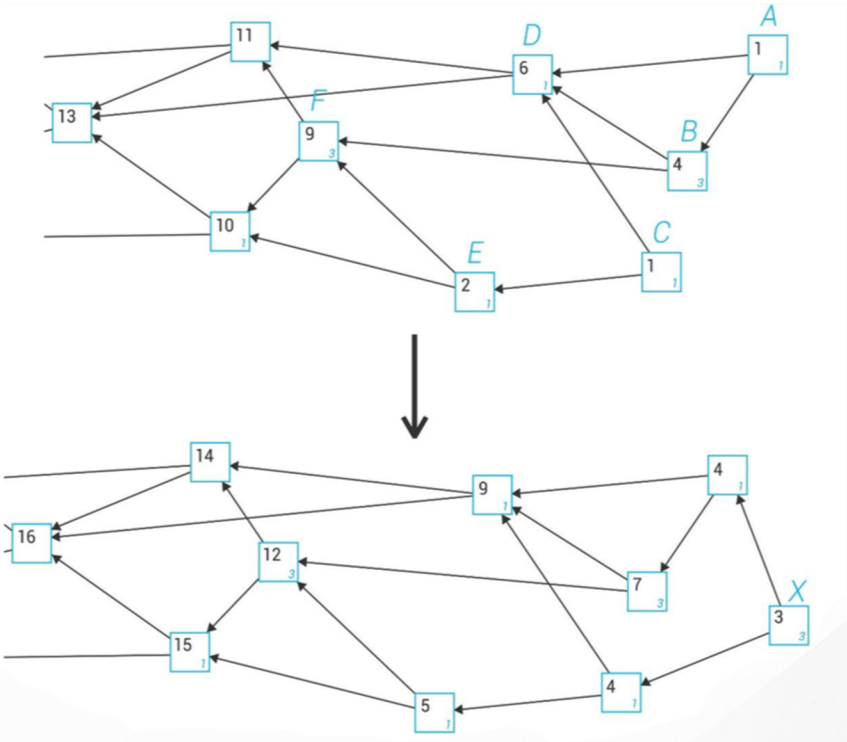
This protection principle assumes, that the user who wants to interact with the system, must first confirm himself.

When you add a new transaction to the graph, it confirms the two past ones. Transactions for confirmation are selected by a certain algorithm, which checks whether these transactions are not contredict and whether they are not accept conflict transactions. For further use, the proof of work is similar to that of Adam Beck's HashCash. The work on proving the reliability of transactions involves scanning to a value that, when hashed using the digest384 algorithm, starts with a certain number of zero trithes.

The algorithm for calculating the total weight can be seen in the figure. Each node (square) is a transaction, the number at the bottom is the transaction's own weight, the number allocated by the bold-cumulative weight. In Figure 2, transaction F is directly or indirectly confirmed by transactions A, B, C, E. The total mass F is 9 (1 + 3 + 1 + 1 + 3). Transactions A, C are the ends of the graph. X, in the second picture, as they indirectly confirm, 3.

The hash is found and the weights are counted on the one with which the transaction was transferred, thereby the device itself is a miner.

In order for users to have an incentive to actively use the Hive, there is a rating system. Having counted the current ratings of the node, you can identify its activity, and depending on this, give more, or store preferences in confirming the transactions of this node. It works like this: a node does a certain job when making or confirming a transaction. It can be said that this work is recorded in the Hive in the form of transactions. Given the time stamps of each transaction, you can create an algorithm that will calculate the current node rating. Here is an example of one of such algorithms. Take the set of all the transactions of the node for a period of time. We assume that for one transaction the node rating is increased by C, whereas for one transaction, the rating is reduced by F. Note that the rating cannot be negative. Let C be the number of the rating for the transaction, D is the ordered set of such elements t-tc, including the element 0, where t is the transaction timestamp (in days), tc is the current timestamp (in days). Then the node's rating at the current time.



Example of adding a new transaction