

# 1. Non-interactive time-based proof of stake finality

Giovanni Antino, Iris Dimni per AiliA SA

## 1.1 Requirements

The protocol must have be linear. Lets suppose that 100 nodes have to coordinate their state, they must do so with the smallest number of messages being sent, in the best case scenario, one per node. Each node must decide independently to accept or refuse a transaction. Control of the network must be distributed as much as possible.

Tolerance to the actions of malevolent actors must reach up to 50%+1 of the total stake so as to approximate the security levels reached by "Proof of Work" algorithms.

## 1.2 Actors involved

An actor is a person, organization, or external system that plays a role in one or more interactions in the system.

### 1.2.1 Mining Nodes

The servers making up the blockchain network are called mining nodes. They are responsible for voting on smart contracts, assigning rewards, selecting, validating and including the transactions into blocks, thus creating the blocks that make up the blockchain.

### 1.2.2 Replicate Nodes

These are servers that help propagate transactions, while keeping a complete copy of the blockchain. They validate blocks and transactions they receive and spread them to the rest of the network but do not create blocks themselves.

### 1.2.3 Holder

Token owners, tokens are needed to operate on the chain.

### 1.2.4 Stake Holder

Address of the tokens owned, necessary to elect a reply node in a mining node. In our case the **TKG**.

### 1.2.5 Green Token - TKG

Tokens necessary for control of the blockchain and usage, similarly to Cardano's Ada they allow:

- Payment of the transactions gas, meaning the fee necessary to conduct a transaction or execute a smart contract.
- Staking on nodes of the network.

It constitutes the main token of the blockchain and is absolutely necessary to make it work. It is created as a result of mining in each block.

### 1.2.6 Red Token - TKR

Token used for payment. They can be used interchangeably with the **TKG** to pay the execution cost of transactions or smart contracts. When the calling address owns a balance of both **TKR** and **TKG** the **TKR** are used first, until the address's balance is emptied, then the **TKG** are used.

All **TKR** are declared in the zero block of the chain using a reserved transaction that can be used exclusively in the same block and is considered invalid afterwards. Owning these tokens does not enable to staking, thus owning them does not confer any kind of control on the blockchain. This Proof of Stake algorithm offers a strong incentive to not use the control tokens, **TKG**, to make payments. The introduction of the **TKR** is one of the solutions to this problem, being an alternative offered to incentivise making use the blockchain while not handing out the control tokens. If this alternative was not present, in those block-chains with an intensive usage the artificial starvation situation would knock on the door and would penalise the usage of the same network.

## 1.3 Network Configuration

The addresses of the nodes that take part actively in the construction of the blockchain are divided in two categories:

- $\mathcal{M}$  Main: Addresses that have no correspondent physical server. They function in the same way as the names of the DNS system, they serve as targets for the stakes placed by the stake holders.
- $\mathcal{O}$  Overflow: Addresses that correspond to physical servers, who do the actual mining. These are the mining nodes.

Stakeholders delegate their voting power to a  $\mathcal{M}$  address using a specific transaction of staking. This stake, or vote, is valid until the same owner decides to abort it using a stake undo action. By staking on a  $\mathcal{M}$ , the power to vote, expressed in the amount of tokens specified, is delegated to the pool of servers represented by the  $\mathcal{M}$ . To be a valid target for staking, a  $\mathcal{M}$  needs to assign at least 1  $\mathcal{O}$  node to itself. A  $\mathcal{M}$  can have an unlimited number of  $\mathcal{O}$  assigned to it. An  $\mathcal{O}$  node on the other hand can only be assigned to a single  $\mathcal{M}$  at a time.

## 1.4 Time scanning and Consent Algorithm

To be able to explain as clearly as possible the concepts expressed in this section we will utilize the default values used in the configuration of the blockchain **Takamaka**.

### 1.4.1 Slot

It's the smallest time unit, corresponding to a time period of 30 seconds. For each slot, the Consensus algorithm chooses a single miner. This is the only one enabled to create a block in that specific time window.

### 1.4.2 Epoch

Slot aggregation, in the default configuration 24000 slots correspond to an epoch. In a best case scenario this will correspond to 24000 blocks being generated.

## 1.5 Slot Assignment

On each Epoch an evaluation is made to distribute the slots of the next epoch between the mining nodes. This evaluation is linked to the stake assigned to each of the  $\mathcal{M}$  they are assigned to. The evaluation takes place on multiple phases and is triggered on the 2/3 completion of the current epoch. All the following steps must be strictly deterministic<sup>1</sup>:

- determine the block with an index number greater than or equal to  $(\text{slotPerEpoch}/3)$  df<sup>2</sup>  $24000/3 = 8000$ , because the blocks are numbered starting from 0 the 7999 or the very next predecessor.
- the seed of this block, concatenated to seeds of an arbitrary but pre-established number of predecessors is passed to a hashing function. The result will generate a ticket (a bit sequence).
- an interval of sequences proportional to the stake assigned to each  $\mathcal{O}$ , according to the stake of the  $\mathcal{M}$  it's assigned to. These intervals will be chained together and produce a space between  $[0, \text{totalStake} - 1)$ <sup>3</sup>
- for each slot to be assigned we will proceed by calculating a number of repetitions of the hash function equal to its *index* + 1. The obtained number modulated to the total stake will fall into one of the concatenated intervals of the previous step. The slot will be assigned to the  $\mathcal{O}$  "owner" of that interval.

## 1.6 Evaluation of the weight of a block sequence

In Bitcoin, the number of linked blocks weighed on the difficulty of the hash is the determining factor in establishing which blockchain to extend. We decided to follow a similar approach in Takamaka.

<sup>1</sup>the detailed explanation of the distribution algorithm, goes beyond the aim of this short introduction

<sup>2</sup>df = refers to default parameters

<sup>3</sup>Correctly assigned to a main with a respective overflow

The nodes decide on which sequence of blocks to extend using the weight of blocks preceding them in the chain. The weight of a block is determined by the total stake of the epoch it belongs to, divided by the number of blocks belonging to that epoch<sup>4</sup>. The mining nodes always extend the chain that has the greater weight.

## 1.7 Decisions on the Purpose

Let's define the following objects:

$\mathcal{T}_{SpE}$  Total Slots per Epoch, df 24000.

$\mathcal{O}_{Stake}^j$  stake assigned to the overflow with index j in an epoch

$\mathcal{T}_{Stake}$  Total Stake accepted per epoch

$\mathcal{O}_{Slots}^j$  slots assigned by the heuristic to the overflow that has index j

$\mathcal{S}_{Weight}$  Weight of a slot as calculated in the previous chapter

$\mathcal{W}_{Oj}^i$  Exact weight that should carry the slot with index i assigned to the overflow of index j considering its stake<sup>5</sup>.

$\mathcal{N}(\mathcal{O})$  total number of overflows

$\mathcal{W}_h^k$  sum of weights of slots from h to k

$W51_h^k$  given an interval of slots  $[h, k]$  this value shows 50% + 1 of the maximum weight reachable in this interval.

$\mathcal{WB}_h^k$  sum of block weights<sup>6</sup> from h to k

$$\mathcal{W}_{Oj}^i = \frac{\mathcal{T}_{SpE} \frac{\mathcal{O}_{Stake}^j}{\mathcal{T}_{Stake}}}{\mathcal{O}_{Slots}^j} \quad (1.1)$$

$$W51_h^k = \frac{\mathcal{WB}_h^k}{(k - h)} \left( \frac{k - h}{2} + 1 \right) \quad (1.2)$$

The algorithm proposed to decide the finality in Takamaka is based on the sliding windows concept, similar to that of tcp/ip. Assuming we have a limited network up to 400 nodes<sup>7</sup> the block of index  $n$  is defined as final if:

$$\mathcal{WB}_{n+1}^{n+400} = \sum_{i=n+1}^{n+400} \left( \sum_{j=0}^{\mathcal{N}(\mathcal{O})} \mathcal{W}_{Oj}^i \right) \quad (1.3)$$

<sup>4</sup>if the epoch is 24000 blocks and the accepted stake is 99000000 **TKG** the weight of a block will be  $99000000/24000 = 4125$

<sup>5</sup>this is necessary to compensate the errors introduced from the heuristic algorithm used to assign the slots

<sup>6</sup>the Takamaka blockchain allows for empty slots in cases such as when an  $\mathcal{O}$  does not send a block in the allotted time window, or when a block contains errors and is discarded.

<sup>7</sup>that is reached admitting the  $\mathcal{M}$  addresses, configured correctly, and summing these stakes and admitting only those that exceed or are at the threshold of 1/400. The techniques used to avoid a stall, when no one reaches this threshold will not be part of this paper.

The sum of the weight of the blocks produced after  $n$  in the range  $(n+1, n+400)$ , denoted as,  $\mathcal{WB}_{n+1}^{n+400}$ , is bigger than or equal to the lower limit of weight for the same interval  $W51_h^k$

$$\mathcal{WB}_{n+1}^{n+400} \geq W51_h^k \quad (1.4)$$

Otherwise i need to extend the interval  $[h, k]$  and move forward making a new evaluation.



## 2. Path finality

Giovanni Antino, Iris Dimni per AiliA SA

### 2.1 Definition of path

A **Path** is any contiguous sequence of blocks, intuitively it is possible to construct a sequence, which includes all the blocks in the path, starting from the newest to the oldest. Within a **Path**, sorting can be established from the following considerations:

- each **Block** is assigned to a **Slot** that we will denote by  $\mathcal{B}_{slot}$
- each **Block** can belong only to one **Epoch** that we will denote by  $\mathcal{B}_{epoch}$
- in a valid sequence of **blocks** each **Slot** can correspond to at most one **Block**
- empty **Slot** may exist.
- each **Block** in a valid sequence is uniquely identified by the value pair  $(\mathcal{B}_{epoch}, \mathcal{B}_{slot})$
- distance, as the absolute number of **Slot** from **Block Zero**, is defined as **Absolute Block Number**. The **Absolute Block Number** of a **Block** is given by  $\mathcal{T}_{SpE}\mathcal{B}_{epoch} + \mathcal{B}_{slot}$

From the above, a strict ordering of the blocks can be established using their **Absolute Block Number**.

#### 2.1.1 Contiguity of two blocks

Given two distinct blocks with **Absolute Block Number**  $k < h$   $b_k$  is contiguous to  $b_h$  if the **Previous Block Hash** of  $b_h$  is equal to the **Single Inclusion Block Hash** of  $b_k$ .

#### 2.1.2 Contiguity of blocks within a Path.

Given the block contiguity relationship, defined in the previous section, we can construct a path as any set of totally ordinate blocks. These sets of blocks define a chain <sup>1</sup>.

#### 2.1.3 Recursive definition of Path

In this section we will define the steps required to create a **Path**.

---

<sup>1</sup>The term chain is sometimes defined as a synonym for a totally ordered set, but it is generally used for referring to a subset of a partially ordered set that is totally ordered for the induced order.

- $\mathcal{P}$  set of paths defined on the blockchain, each **Path** is associated with an integer.
- $|\mathcal{P}|$  Number of **Path** defined within  $\mathcal{P}$  (cardinality of  $\mathcal{P}$ ).
- $p_n$   $n$ -th **Path** belonging to  $\mathcal{P}$ , each **Path** is characterized by an index and an ordered list of blocks associated with it.  $p_n = (n, \langle b_h, \dots, b_k \rangle)$

### 2.1.3.1 Step 0

I define  $p_0$  as the first **Path** associated with the  $b_0$  of the blockchain.  $p_0 = (0, \langle b_0 \rangle)$

### 2.1.3.2 Step n

I add  $b_z$  to the blockchain, proceed to evaluate **Path** for  $b_z$ .

- If **Previous Block Hash** of  $b_z$  does not match the **Single Inclusion Block Hash** of a block in the blockchain  $b_z$  cannot be linked and therefore evaluated.  $b_z$  is not assigned to any **Path**.
- If **Previous Block Hash** of  $b_z$  corresponds to the **Single Inclusion Block Hash** of a block in the blockchain  $b_z$ :
  - we obtain  $b_k$  where **Single Inclusion Block Hash** of  $b_k = \text{Previous Block Hash}$  of  $b_z$
  - if  $b_k$  does not belong to any **Path** I stop the procedure
  - If  $b_k$  belongs to a **Path**, which we will call  $p_k$  for convenience, and  $b_k$  is the last **Block** in the chain of  $p_k$  blocks then I add  $b_z$  as the last **Block** of  $p_k$ . Es.  $p_k^n = (k, \langle b_i, \dots, b_k \rangle) \rightarrow p_k^{n+1} = (k, \langle b_i, \dots, b_k, b_z \rangle)$
  - if  $b_k$  belongs to a **Path**, which we will call  $p_k$  for convenience and  $b_k$  is not the last **Block** in the chain of  $p_k$  blocks then I create a new **Path**, for convenience I will call it  $p_z = (||| + 1, \langle b_z \rangle)$  where the index will be the current cardinality of the set  $\mathcal{P} + 1$  and  $b_z$  will be the only element of the new  $\mathcal{P}$ .

## 2.2 Path weight

Applying the definition of path given in 1.6 to a **Path** we can determine which of its constituent blocks are final. We now extend the definition of the recursive function 2.1.3 by adding the evaluation of the weight of the blocks.

To simplify the notation, instead of indicating slots by the pair  $(\mathcal{B}_{epoch}, \mathcal{B}_{slot})$ , the equivalent notation with only **Absolute Block Number** will be used.

By  $\mathcal{S}_{Weight}(k)$  I denote weight of slot  $k$  in **Absolute Block Number** notation.

### 2.2.0.1 Step 0

I define  $p_0$  as the first **Path** associated with the  $b_0$  of the blockchain.  $p_0 = (0, \langle b_0 \rangle)$ . The block is also associated a weight value  $\mathcal{B}_{Weight}$ . the definition of  $p_0$  becomes  $(0, \langle b_0^{\mathcal{B}_{Weight}} \rangle)$  which, in the case of block zero is  $(0, \langle b_0^{\mathcal{S}_{Weight}} \rangle)$ . I define  $\check{\mathcal{B}}$  as the set of final blocks where  $\check{\mathcal{B}}_0 = \{\emptyset\}$ . I define  $\check{\check{\mathcal{B}}}_0^0$  as the set of nonfinal blocks in the path of  $b_0$  and therefore  $\check{\check{\mathcal{B}}}_0^0 = \{b_0\}$ .

### 2.2.0.2 Step n

I add  $b_z$  to the blockchain, proceed to evaluate **Path** by  $b_z$ .

1. If **Previous Block Hash** of  $b_z$  does not match the **Single Inclusion Block Hash** of a block in the blockchain  $b_z$  cannot be linked and therefore evaluated.  $b_z$  is not assigned to any **Path** and its weight cannot be determined. This terminates the evaluation.
2. If **Previous Block Hash** of  $b_z$  corresponds to the **Single Inclusion Block Hash** of a block in the blockchain  $b_z$ :
  - (a) I obtain  $b_k$  where **Single Inclusion Block Hash** of  $b_k = \text{Previous Block Hash} of  $b_z$ .$
  - (b) If  $b_k$  does not belong to any **Path** I stop the procedure.
  - (c) If  $b_k$  belongs to a **Path**, which we will call  $p_k$  for convenience, and  $b_k$  is the last **Block** in the chain of  $p_k$  blocks then I add  $b_z$  as the last **Blockz** of  $p_k$ . Ex.  $p_k^n = (k, \langle b_i^{\mathcal{WB}_0^i}, \dots, b_k^{\mathcal{WB}_0^k} \rangle) \rightarrow p_k^{n+1} = (k, \langle b_i^{\mathcal{WB}_0^i}, \dots, b_k^{\mathcal{WB}_0^k}, b_z^{\mathcal{WB}_0^z} \rangle)$  dove  $b_z^{\mathcal{WB}_0^z} = b_k^{\mathcal{WB}_0^k + \text{Weight}(z)}$
  - (d) If  $b_k$  belongs to a **Path**, which we will call  $p_k$  for convenience, and  $b_k$  is not the last **Block** in the chain of  $p_k$  blocks then I create a new **Path**, for convenience I will call it  $p_z = (|\mathcal{P}| + 1, \langle b_z^{\mathcal{WB}_0^z} \rangle)$  where the index will be the current cardinality of the set  $\mathcal{P}+1$  and  $b_z$  will be the only element of the new  $\mathcal{P}$  with calculated weight  $b_z^{\mathcal{WB}_0^z} = b_k^{\mathcal{WB}_0^k + \text{Weight}(z)}$ .
3. I proceed with the reassessment of finality. Given  $\ddot{\mathcal{B}}$  set of nonfinal blocks included in a **Path**. From this set I obtain the subset of blocks  $\ddot{\mathcal{B}}_m^n = (b_m, \dots, b_n)$  found in a chain from  $b_z$  a  $b_0$ .  $\forall b_j \in \ddot{\mathcal{B}}_m^n \wedge (z - j \geq \mathcal{N}(\mathcal{O}))$  I calculate  $W5I_j^z$ . Given the set of final blocks  $\check{\mathcal{B}}$  if  $\mathcal{WB}_j^z \geq W5I_j^z$  we update  $\check{\mathcal{B}}$  as  $\check{\mathcal{B}} = \check{\mathcal{B}} \cup b_j$ . We remove  $b_j$  from  $\ddot{\mathcal{B}}$ , therefore  $\ddot{\mathcal{B}} = \ddot{\mathcal{B}} \setminus b_j$ .

## 2.3 Dead Block

Starting from 2.2 I define the dead **Block**. A dead **Block** is a **Block** whose predecessor is connected to a **Block** with a final successor. Suppose we have  $b_n$  belonging to the set of final blocks. Let us take  $b_k$  belonging to the chain of blocks  $\langle b_0, \dots, b_k, \dots, b_n \rangle$ . Clearly  $b_k$  is final given how the definition of finality is constructed (I cannot have a non-final predecessor of a final block). Now suppose we connect a block  $b_i$  to  $b_k$ . If I allow this behavior and the extension of a **Path** connected to  $b_k$  the **Path** that includes  $b_i$  could make  $b_i$  final. In this way I would end up with the blockchain  $\langle b_0, \dots, b_k, \dots, b_i, \dots \rangle$  which does not include  $b_n$  effectively making  $b_n$  nonfinal.

To avoid this paradoxical situation we define dead **blocks**. Given a non-final block  $b_x$ . Suppose that  $b_x$  has an immediate final predecessor  $b_y$ . Suppose that  $b_p$  exists and that  $b_p$  is itself final and has as its immediate predecessor  $b_y$ . By construction  $b_x$  and  $b_p$  are on distinct **Path** (see 2d) since each block can have only one predecessor. At this point I declare  $b_x$  as dead **Block** and all non-final blocks that have  $b_x$  as predecessor in their chain as dead blocks in turn. If a new  $b_p$  block has a dead block as its predecessor the block is discarded and cannot join the blockchain.

In this way I prevent the extension of paths and chains that may invalidate the purpose achieved by a block, in the specific case  $b_n$ .