Sycomore : a Permissionless Distributed Ledger that self-adapts to Transactions Demand

Emmanuelle Anceaume
CNRS / IRISA, France
emmanuelle.anceaume@irisa.fr

Antoine Guellier
CNRS / IRISA, France
antoine.guellier@irisa.fr

Romaric Ludinard
IMT Atlantique, France
romaric.ludinard@imt-atlantique.fr

Bruno Sericola
Inria, France
bruno.sericola@inria.fr

***Abstract*—We propose a new way to organise both transactions and blocks in a distributed ledger to address the performance issues of permissionless ledgers. In contrast to most of the existing solutions in which the ledger is a chain of blocks extracted from a tree or a graph of chains, we present a distributed ledger whose structure is a balanced directed acyclic graph of blocks. We call this specific graph a SYC-DAG. We show that a SYC-DAG allows us to keep all the remarkable properties of the Bitcoin blockchain in terms of security, immutability, and transparency, while enjoying higher throughput and self-adaptivity to transactions demand. To the best of our knowledge, such a design has never been proposed so far.**

I. Introduction

The goal of decentralized cryptocurrency systems is to offer a medium of exchange secured by cryptography, without the need of a centralized banking authority. An increasing number of distributed cryptocurrencies systems are emerging, and among them Bitcoin, which is often designated as the pioneer of this kind of systems. Bitcoin circumvents the absence of a global trusted third-party by relying on a blockchain, an append-only data-structure, publicly readable and writable, in which all the valid transactions ever issued in the system are progressively appended through the creation of cryptographically linked blocks. Construction of the blockchain is distributed among the nodes of the system (several thousand of nodes), and is secure even if malicious nodes own almost the majority of the computational power of the system. This high resilience relies on the smart orchestration of cryptography primitives, distributed algorithms, and incentive mechanisms.

From a chain of blocks to a balanced directed acyclic graph of blocks. In a chain of blocks, each block references an earlier block by inserting a cryptographic link to this earlier block in its header. This forms a tree of blocks, rooted at the genesis block, in which a branch is a path from a leaf block to the root of the tree. Each branch, taken in isolation, represents a consistent history of the crypto-currency system, that is, does not internally contain any conflicting transactions – double-spending transactions. On the other hand, any two branches of the tree do not need to be consistent with each other. The reason is that, at any time, each node of the system selects a unique branch to represent the history of the crypto-system – this branch being the longest one of the tree, or equivalently the one that required the most important quantity of work. Garay and Kiayias [8] have characterized Bitcoin blockchain via its quality and its common prefix properties. Specifically, they have shown that, by keeping the creation rate of blocks very low with respect to their propagation time in the network *(i)* if the adversary controls no more than $1/3$ of the network hashing power then it provably controls less than a majority of the blocks in the blockchain and, *(ii)* if the adversary controls no more than $1/2$ of the network hashing power then the blockchains maintained by any two honest nodes possess a large common prefix (up to the last k appended blocks), and the probability that they are not mutual prefix of each other decays exponentially in k. Unfortunately, this compels Bitcoin's blockchain to a low transaction confirmation throughput: no more than 7 transactions can be permanently confirmed per second in average, which is by too far very low.

A recent evolution in the structure of blockchains is emerging to deal with the performance issue aforementioned. Eyal et al. [7] propose with BITCOIN-NG an off-chain mechanism: blocks refer to a leader in charge of validating transactions batched in micro-blocks out of the chain. LIGHTNING [12] follows the same principle by publishing results of a set of transactions among two parties. HASHGRAPH [2], BYTEBALL [3], and IOTA [13] leverage the presence of well known institutions to get rid of blocks, GHOST [16] and SPECTRE [15] protocols family modifies the blockchain data structure from linked-list of blocks to a directed graph of blocks. In particular, SPECTRE [15] organises blocks in a directed (but not acyclic) graph of blocks. Blocks are built so that they acknowledge the state of the directed graph at the time blocks were created which decreases the opportunity for powerful attackers to create blocks in advance. The number of created blocks is increased with respect to classic blockchain-based systems, since all the concurrently created blocks belong to the directed graph. Unfortunately the absence of mechanisms to prevent the presence of conflictual records (i.e., blocks with conflicting transactions) and the cycles in the directed graph require that participants execute a complex algorithm to extract from the graph the set of accepted (i.e., valid) transactions [15].

SYC-DAG: A balanced directed acyclic graph of blocks. The ledger we propose is not a chain of blocks, or a set of transactions extracted from a tree of chains—as achieved in existing solutions, e.g., [4], [7], [9], [10], [11], [15], [16] – but

a dedicated balanced directed acyclic graph of blocks, that we call in the following a SYC-DAG. Construction of the SYC-DAG ledger is achieved by the Sycomore protocol, a new distributed protocol run by all the nodes of the system. The Sycomore protocol, that we denote in the following by π_{syc}, resembles in some aspects to Nakamoto's one, denoted by π_{nak}. Briefly, π_{syc} is a distributed protocol allowing each participating node to reach an agreement on a SYC-DAG. Similarly to π_{nak}, at the core of π_{syc} are two effective and simple ingredients: a hashcash proof-of-work (PoW) mechanism to synchronize the construction of blocks and to incentive nodes to correctly behave and a block selection rule to handle block concurrency. On the other hand, the unique features of π_{syc} are the following ones.

1) The structure of the SYC-DAG self-adapts to fluctuations of transactions demand: chains of blocks naturally split into multiple ones to cope with burst of pending transactions, while they merge back together to adapt to a drop of activity;
2) The probability of forks decreases with the increasing number of leaf blocks. Thus the rate at which blocks are mined can be drastically augmented (e.g., one block every 20 seconds) without incurring more forks than in Bitcoin;
3) The frequency at which blocks are mined self-adapts to the number of leaf-blocks in the SYC-DAG;
4) Miners cannot predict the branch in the SYC-DAG to which their blocks will be appended. This prevents malicious miners from devoting their computational work on the growing of specific branches of the SYC-DAG.
5) Transactions are dynamically and evenly partitioned over the SYC-DAG, i.e., there is no transaction that belongs to two different blocks. This makes effective the parallelism brought by the SYC-DAG, which allows us to increase the number of confirmed transactions per second.

Altogether, these properties allow us to build a secure, immutable and efficient distributed ledger whose structure self-adapts to transactions demand.

The remaining of the paper is orchestrated as follows. Section II presents the model of the system. Section III introduces the SYC-DAG data structure and describes how π_{syc} builds the SYC-DAG in a permissionless and distributed system. Properties met by π_{syc} are stated in Section IV. Finally Section V concludes.

II. Model of the system

We suppose a *permissionless* system, that is a system populated by a large, finite yet unbounded set of nodes and whose composition may change over time, which participate to the construction of the distributed ledger. The communication delays between any two nodes, the time to execute a local computation step, and the drift of local clocks are assumed to be upper-bounded, however these upper-bounds are unknown to nodes. These temporal assumptions fit the semi-synchrony model [6]. To fit the permissionless model, we assume the *computational threshold adversary model*, which describes the power of the adversary in terms of computational resources it controls [8]. Note that in the more classical distributed computing literature, the *threshold adversary model* describes the power of the adversary in terms of the number of nodes it manipulates. Finally, as we are focusing on a financial cryptosystem, we cannot just consider that nodes are either obedient (*i.e.*, they follow the prescribed algorithm) or malicious. We suppose that most of the nodes are rational, that is, strategically behave to increase their own utility function without violating the prescribed protocols. For example, a rational miner may in priority insert in its block all the current transactions that provide the maximal fees while an obedient one will insert transactions irrespective of the gain they procure. Since rational nodes do not violate the protocol specification, in the following we consider as correct both obedient and rational nodes. We assume that nodes have access to basic and safe cryptographic functions, including a cryptographic hash function $\mathfrak{h}$ – modeled as a random oracle – and an asymmetric signature scheme – that allows a node r to generate public and secret key pairs (p_r, s_r), to compute signatures $\sigma_{r,\mathfrak{h}(d)}$ on messages d, and to verify the authenticity of a signature. By these properties, each object o of the system – *i.e.*, transaction and block – is assumed to be uniquely identified. In practice, this is achieved by computing o's cryptographic digest $\mathfrak{h}(o)$. We assume that correct nodes use their cryptographic keys correctly, *i.e.* they do not disclose, share or drop their secret keys. Finally, we do not suppose the existence of any trusted public key infrastructure (PKI) to establish nodes identities.

III. The Sycomore Protocol

The objective of the Sycomore protocol π_{syc} is to allow each node u of the system to locally maintain its own local view $\mathcal{L}_u$ of the ledger, so that at any time u can extract from $\mathcal{L}_u$ a SYC-DAG (defined in Section III-A) denoted by $\mathcal{L}_u^\star$ such that for any two nodes u and v either $\mathcal{L}_u^\star \sqsubseteq \mathcal{L}_v^\star$ or $\mathcal{L}_v^\star \sqsubseteq \mathcal{L}_u^\star$.

We assume a "Bitcoin like crypto-system". That is, similarly to π_{nak}, users create transactions to buy goods with coins. A transaction T is uniquely identified by the hash of its content. Its content is made of two sets, the input set denoted by I and the output one denoted by O. Set I contains a set of references to UTXOs (Unspent Transaction Outputs), credited by previous transactions, together with the proof that the creator of T is allowed to redeem each of those UTXOs. The output set O contains a set of "accounts" (or addresses in the Bitcoin terminology), together with the amount of coins to be credited and the challenges that will allow their owners to redeem these coins. Once created, users submit their transactions to the peer-to-peer network. Each node of the network should check the validity of each received transaction prior to propagating it to its neighborhood. Informally, a transaction $T = (I, O)$ is locally valid at node p if p has received all the transactions that have credited all the inputs in I and has never received transactions already using any of those inputs. If there exists some transaction $T' = (I', O')$ and some input i such that i belongs to both I and I', then input i is said to be in a *double-spending situation*. Transaction $T = (I, O)$ is *conflict-*

free if none of the inputs of T is involved in a double-spending situation and all of the transactions that credited T's inputs are conflict-free. By construction, the induction is finite because Bitcoin creates money only through coinbase transactions, which are by definition conflict-free [1]. Similarly to π_{nak} algorithm, blocks are created by successful miners, a subset of the nodes involved in the proof-of-work competition. The incentive to participate to this competition is provided by a reward given to each successful miner. This reward is made of a fixed amount of coins, and a fee associated to each transaction contained in the newly created block. This reward is inserted in the output of a particular transaction, called the coinbase transaction.

To fully take advantage of the SYC-DAG data structure, blocks content is slightly different from the way blocks are built in π_{nak}. Briefly, blocks header contain a commitment for each leaf block of the SYC-DAG (in Bitcoin, the commitment is reduced to a single leaf block), and the choice of the set of transactions embedded in the blocks takes advantage of the presence of several parallel chains in the SYC-DAG. Prior to providing a detailed description of the set of information contained in a block, and how blocks are locally inserted in the local view of the ledger, we first introduce the notion of SYC-DAG.

*A. A Sycomore Directed Acyclic Graph (*SYC-DAG*)*

Let us consider graph $G = (V, E)$ defined by a set of vertices V and a set of edges E. Vertices are blocks of transactions and edges represent a predecessor relationship. More precisely and similarly to Bitcoin, a block $b_j \in V$ "points" to a previous existing block $b_i \in V$, that is, edges point back in time. We have $(b_j, b_i) \in E$, nevertheless we say that b_i is the predecessor of b_j and thus b_j is the successor of b_i. Given two blocks b_i and $b_j \in V$, we say that b_i is reachable from b_j if there exists a path in G from b_j to b_i. We denote $b_j \rightsquigarrow b_i$. By convention, b_i is always reachable from itself, *i.e.,* $b_i \rightsquigarrow b_i$.

A *leaf block* refers to a block $b_i \in V$ with no children (i.e., block b_i is such that $\{b_j \in V \mid b_j \rightsquigarrow b_i\} = \{b_i\}$). Conversely, a *genesis block* is a block $b_i \in V$ which is reachable from all the blocks in G, (i.e., we have $\{b_j \in V \mid b_j \rightsquigarrow b_i\} = V$). In the following, a genesis block is denoted by b_0.

We define $G' = (V', E')$ as a *prefix* of $G = (V, E)$, denoted by $G' \sqsubseteq G$, if $V' \subseteq V$ and $E' \subseteq E$ and for any edge $(b_j, b_i) \in E$, we have $b_j \in V' \Rightarrow [b_i \in V' \wedge (b_j, b_i) \in E']$. Note that b_0 belongs to all the prefixes of G.

The notion of *chain* is central to our construction.

Definition 1 (Chain)**.** *A chain $\mathcal{C} = (b_1 b_2 \dots b_c)$, with $c \geq 2$, in G is a sequence of blocks starting at block b_1 and ending at block b_c such that*

- *each block b_i, with $2 \leq i \leq c$ in the sequence points to a unique block and this block belongs to the sequence and,*
- *each block b_i, with $1 \leq i \leq c-1$ in the sequence is pointed by a unique block and this block belongs to the sequence, and*
- *the first block b_1 does not point to any block of the sequence and the last block b_c is not pointed by any block in the sequence.*

The length of a chain is the number of blocks of this chain.

Each block in G is assigned a binary string called *label*. The length of a label ℓ is the number of bits of ℓ, and is denoted by $|\ell|$. A genesis block b_0 is labelled with the empty binary string ε. All the blocks of a chain have the label of the first block of the chain they belong to. Thus, all the blocks of the chain starting with the genesis block are labelled with the empty string ε. Henceforth, a block will refer to both the block and its label. We use the following notations in the remaining of the paper: $\mathcal{C}^\ell$ denotes a chain $\mathcal{C}$ whose label is ℓ, b^ℓ denotes a block whose label is ℓ, and b_i^ℓ indicates that block b is the i-th block, $i \geq 1$, of a chain labelled with ℓ. Finally, the binary string of label ℓ of length m is noted $\ell_0 \ell_1 \dots \ell_{m-1}$, and we denote by $\ell^{\lfloor k}, 0 \leq k < m$, the prefix of ℓ of length k, i.e., the k first bits of ℓ.

The load of a block is the ratio between its number of bytes and its maximal load (for instance, $1\,\text{MB}$ in Bitcoin prior to the date of SegWit activation). The load of a block is returned by function LOAD(). We introduce the following three system parameters: integer $c_{\min} > 0$, and thresholds $0 < \gamma < \Gamma < 1$ under and above which a block is considered respectively under and upper loaded.

Definition 2 (Splittable and Mergeable Block)**.** *Let $\mathcal{C} = (b_1 b_2 \dots b_c)$ be a chain with $c \geq c_{\min}$, then*

- *b_c is called a splittable block if*
$$\frac{1}{c_{\min}} \sum_{j=1}^{c_{\min}} \text{LOAD}(b_{c-c_{\min}+j}) > \Gamma.$$
- *b_c is called a mergeable block if*
$$\frac{1}{c_{\min}} \sum_{j=1}^{c_{\min}} \text{LOAD}(b_{c-c_{\min}+j}) < \gamma.$$

A block which is neither splittable nor mergeable is called a regular block.

We will rely on a *distance* function $\mathcal{D}$ that will allow us to uniquely characterize the closest label to a given bit string. This is obtained by computing the numerical value of the "exclusive or" (XOR) of bit strings. To prevent two bit strings to be at the same distance from a given one, bit strings are suffixed with as many bits "0" as needed to equalize their size to the maximal number s of bits of the longest bit string. Suffixing a bit string ℓ with one "0" is denoted by $\ell.0$.

Definition 3 (Distance $\mathcal{D}$)**.** *Let $a = a_0 \dots a_{d-1}$ and $b = b_0 \dots b_{d'-1}$ be any two bit strings (note that the bit numbering starts at zero for the most significant bit), and $s = max(d, d')$.*

$$\begin{aligned} \mathcal{D}(a,b) &= \mathcal{D}(a_0 \dots a_{d-1}.0^{s-d}, b_0 \dots b_{d'-1}.0^{s-d'}) \\ &= \sum_{i=0}^{s-1} 2^{s-1-i} \mathbf{1}_{\{a_i \neq b_i\}}, \end{aligned}$$

where notation $\mathbf{1}_{\{A\}}$ denotes the indicator function which is equal to 1 if the set A is not empty and 0 otherwise. Hence, bit

strings that have longer prefix in common are closer to each other. For instance, let $\ell = 0001$, $\ell' = 0000$, and $\ell'' = 111$ be any three bit string labels, then $\mathcal{D}(\ell, \ell') = 1$, $\mathcal{D}(\ell, \ell'') = 8 + 4 + 2 + 1 = 15$ and $\mathcal{D}(\ell', \ell'') = 8 + 4 + 2 = 14$. Two binary strings at distance 1 are called siblings, e.g. bit strings ℓ and ℓ' are sibling, which is denoted by $\ell = \overline{\ell'}$. Note that by abuse of notation, two sibling blocks b and b' will be denoted by b and $\overline{b'}$.

Definition 4 (Split Chains)**.** *Chain $\mathcal{C}^{\ell^{(1)}} = (b_1^{\ell^{(1)}} \dots b_n^{\ell^{(1)}})$, $n \geq 1$, and chain $\mathcal{C}^{\ell^{(2)}} = (b_1^{\ell^{(2)}} \dots b_m^{\ell^{(2)}})$, $m \geq 1$ are split chains if $b_1^{\ell^{(1)}}$ and $b_1^{\ell^{(2)}}$ point back to the same splittable block b^{ℓ}. We have $\ell^{(1)} = \ell.0$ and $\ell^{(2)} = \ell.1$.*

Note that by definition of the labels, all the blocks of $\mathcal{C}^{\ell^{(1)}}$ (resp. $\mathcal{C}^{\ell^{(2)}}$) have labels $\ell^{(1)}$ (resp. $\ell^{(2)}$).

Definition 5 (Merged Chain)**.** *Chain $\mathcal{C}^{\ell} = (b_1^{\ell} \dots b_j^{\ell})$, $j \geq 1$, is a merged chain if b_1^{ℓ} points back to two sibling mergeable blocks $b^{\ell^{(1)}}$ and $b^{\ell^{(2)}}$ (i.e., $b^{\ell^{(1)}}$ and $b^{\ell^{(2)}}$ are two mergeable blocks whose respective chains are issued from the same splittable block) such that $\ell^{(1)} = \ell_0 \dots \ell_{|\ell^{(1)}|-1}$, $\ell^{(1)} = \overline{\ell^{(2)}}$ and we have $\ell = \ell_0 \dots \ell_{|\ell^{(1)}|-2}$.*

By abuse of language, we say that chain $\mathcal{C}^{\ell}$ is the predecessor of chain $\mathcal{C}^{\ell'}$ if the first block of $\mathcal{C}^{\ell'}$ points back to the last block of $\mathcal{C}^{\ell}$. This is denoted by $\mathcal{C}^{\ell} = \text{pred}(\mathcal{C}^{\ell'})$.

Based on the above definitions, we are ready to define what is a SYC-DAG.

Definition 6 (SYC-DAG)**.** *A graph $G = (V, E)$ is a* SYC-DAG *if G has a unique genesis block b_0 and there exists a partition $\mathcal{P} = \{\mathcal{C}^{\ell_1}, \dots, \mathcal{C}^{\ell_n}\}$ of V such that $\forall i$ s.t. $1 \leq i \leq n, \mathcal{C}^{\ell_i}$ is a chain with label ℓ_i (note that several chains in $\mathcal{P}$ may be assigned the same label) and the following three properties hold:*

$$\forall \mathcal{C}^{\ell_i} \in \mathcal{P}, \forall k \text{ s.t. } 0 \leq k < |\ell_i|, \mathcal{C}^{\ell_i^{\lfloor k}} \in \mathcal{P} \quad (1)$$

$$\forall \mathcal{C}^{\ell_i} \text{ a merged chain } \in \mathcal{P}, \mathcal{C}^{\ell_i.0}, \mathcal{C}^{\ell_i.1} \in \mathcal{P} \quad (2)$$

$$\forall \mathcal{C}^{\ell_i}, \mathcal{C}^{\ell_j} \in \mathcal{P}, \ell_i = \ell_j \Rightarrow [\text{pred}(\mathcal{C}^{\ell_i}) \neq \text{pred}(\mathcal{C}^{\ell_j})] \quad (3)$$

Note that if G is a SYC-DAG then by Property 1, $\mathcal{C}^{\varepsilon} \in \mathcal{P}$. Figure 1 shows a possible ledger built by π_{syc}, and we will use it to illustrate all introduced notions. Vertices of graph $G = (V, E)$ are partitionned into $\mathcal{P} = \{\mathcal{C}_1^{\varepsilon}, \mathcal{C}_2^0, \mathcal{C}_3^1, \mathcal{C}_4^{00}, \mathcal{C}_5^{01}, \mathcal{C}_6^{10}, \mathcal{C}_7^{11}, \mathcal{C}_8^0, \mathcal{C}_9^{00}, \mathcal{C}_{10}^{01}\}$. For any chain $\mathcal{C}^{\ell} \in \mathcal{P}$ we have $\forall k$ s.t. $0 \leq k < |\ell|, \mathcal{C}^{\lfloor k} \in \mathcal{P}$. In particular, we have $\mathcal{C}^{\varepsilon} \in \mathcal{P}$. For any chains $\mathcal{C}_i^{\ell}, \mathcal{C}_j^{\ell}$ (*e.g.* $\mathcal{C}_2^0$ and $\mathcal{C}_8^0$), we have $\text{pred}(\mathcal{C}_i) \neq \text{pred}(\mathcal{C}_j)$. Finally, for any merged chain $\mathcal{C}_i^{\ell} \in \mathcal{P}$ (e.g. $\mathcal{C}_8^0$), we have b_1^{ℓ} points back to two sibling mergeable blocks $b_j^{\ell.0}$ and $b_k^{\ell.1}$ (i.e., last blocks of respectively $\mathcal{C}_4^{00} \in \mathcal{P}$ and $\mathcal{C}_5^{01} \in \mathcal{P}$).

Remark that most of the blockchain-like systems build a chain of blocks, which is a restriction of a SYC-DAG to a unique chain.

B. Construction of a block

We are now ready to describe how blocks are constructed to allow each node of the system to locally maintain its local view of the ledger. As previously said, blocks contain additional information with respect to Bitcoin's blocks so that *(i)* malicious miners cannot devote their computational power to the growing of a specific chain of the SYC-DAG (Property 1), *(ii)* transactions are partitioned over the blocks of the SYC-DAG (Property 2), and *(iii)* the number of leaf blocks self-adapts to transactions demand (Property 3).

Property 1. *The predecessor of a block is neither predictable nor choosable.*

Let u be any miner, b be the block miner u is currently creating, $\mathcal{L}_u$ be u's local view of ledger, and $b^{\ell_1} \dots b^{\ell_j}$ be the leaf blocks of $\mathcal{L}_u$. The objective of Property 1 is to guarantee that miner u can neither foresee nor chose the chain to which its block will be appended (if it is effectively appended) prior to having irremediably completed the construction of his block's header. The reason is to prevent an adversary from devoting all its computational power to the growing of a specific chain. The solution we propose to achieve this goal is *(i)* to insert in b_u's header the view on the current state of $\mathcal{L}_u$ (i.e. hashes of the leaf blocks), and *(ii)* a commitment on the future state of $\mathcal{L}_u$, that is a commitment that must be valid whatever the leaf block to which b_u will be appended. As detailed below, this commitment is sealed with a proof-of-work. Finally, all these pieces of information allows us to derive the predecessor of b_u. Specifically, when u creates block b_u, it inserts in b_u's header a set $\mathcal{H}$ defined as

$$\mathcal{H} = \left\{ \left(\mathfrak{h}(b^{\ell_1}), \ell_1', m^{\ell_1'} \right), \dots, \left(\mathfrak{h}(b^{\ell_j}), \ell_j', m^{\ell_j'} \right) \right\}, \quad (4)$$

where $\mathfrak{h}(b^{\ell_1}) \dots \mathfrak{h}(b^{\ell_j})$ are the hashes of the leaf blocks $b^{\ell_1} \dots b^{\ell_j}$ of $\mathcal{L}_u$, and for each b^{ℓ_i}, $1 \leq i \leq j$, label ℓ_i' represents the label of the next block to be appended to b^{ℓ_i}. That is if b^{ℓ_i} is a regular block, then by construction of the SYC-DAG, $\ell_i' = \ell_i$. On the other hand, if b^{ℓ_i} is a splittable block then by Definition 4, only two split chains can succeed to block b^{ℓ_i}, and thus two tuples $\left(\mathfrak{h}(b^{\ell_i}), \ell_i.0, m^{\ell_i.0}\right)$ and $\left(\mathfrak{h}(b^{\ell_i}), \ell_i.1, m^{\ell_i.1}\right)$ will appear in $\mathcal{H}$. Finally, if b^{ℓ_i} is a mergeable block and its sibling leaf block is also mergeable, then by Definition 5, only a merged chain can succeed to block b^{ℓ_i}, and thus two tuples $\left(\mathfrak{h}(b^{\ell_i}), \ell_i', m^{\ell_i'}\right)$ and $\left(\mathfrak{h}(b^{\overline{\ell_i}}), \ell_i', m^{\ell_i'}\right)$ will appear in $\mathcal{H}$, with ℓ_i' the largest common prefix of ℓ_i and $\overline{\ell_i}$. Finally, for each label ℓ_i', the Merkle root $m^{\ell_i'}$ of the set of locally pending transactions whose identifier is prefixed by ℓ_i' is inserted in $\mathcal{H}$ (Property 2 will detail this point). Hence, if j is the number of leaf blocks in $\mathcal{L}_u$ (at the time miner u created b_u), then the number of tuples in $\mathcal{H}$ is equal to $j + j_1$ if j_1 leaf blocks of $\mathcal{L}_u$ are splittable.[1]

Now miner u derives from $\mathcal{H}$ the leaf block that will be the predecessor of b_u without being able to favor one leaf block among all the leaf blocks of $\mathcal{L}_u$, while guaranteeing this choice to be verifiable by anyone. This is achieved by relying on the proof-of-work mechanism. Specifically, miner u works to find a nonce ν such that $\mathfrak{h}(\mathcal{H}||\nu) \leq T$, where T is

[1]Note that as long as a splittable block has not been appended by two split blocks then that splittable block is still considered as a leaf block.

the target of the proof-of-work, and —— is the concatenation operator. The difficulty of obtaining a hash that matches the specific pattern T increases exponentially as the number of the most significant bits in T set to "0" increases. Once nonce ν is found, miner u inserts ν in b_u's header, and derives the unique predecessor of his block b_u as the leaf block (let us call it b^{ℓ_i}) whose successor will carry the closest label ℓ'_i to $b = \mathfrak{h}(\mathcal{H}||\nu) \bmod 2^s$, i.e.,

$$\ell'_i \quad = \quad \underset{\ell'_k \text{ s.t. } (-,\ell'_k,-)\in\mathcal{H}}{\arg\min} \quad \mathcal{D}(\ell'_k, b),$$

where s is the number of bits of the longest successor's label, and the distance $\mathcal{D}$ is computed as defined in Definition 3. The label ℓ_u of b_u is set to ℓ'_i. Notice that in case ℓ'_i belongs to two tuples in $\mathcal{H}$, then block b_u will have as predecessor the two mergeable blocks $b_i^{\ell_i}$ and $b_i^{\overline{\ell_i}}$.

It is very important to see that no specific reference to the predecessor is added in a block header. The reason is that the header is sealed (thanks to the proof-of-work) prior to having derived such an information. As a consequence, when a node receives a block, it checks the block validity, and derives the block predecessor.

In the following we suppose the absence of temporary forks. Their resolution is detailed in Section III-C.

Property 2. *For any transaction $T \in \mathcal{L}_u$,*

- *it exists a unique block $b^\ell \in \mathcal{L}_u$ such that $T \in b^\ell$, and*
- *it exists a unique path starting from a leaf block $b^{\ell'}$ to the genesis block that contains b^ℓ, and both ℓ' and ℓ share a common prefix.*

Property 2 is related to the parallel creation of blocks. It guarantees that a transaction belongs to a unique block, and that looking for that block in the ledger only requires to traverse a single and well identified path whose length is in the best case logarithmic in the number of blocks of the ledger. As briefly aforementioned, the solution we propose to implement Property 2 is to partition pending transactions according to the prefix of their identifier so that only transactions that are prefixed by label ℓ_u are inserted in block b_u (see Property 1). Note that the Merkle root of this set of pending transactions exactly matches the Merkle root that appears in tuple $\left(b_i^{\ell_i}, \ell'_i, m^{\ell'_i}\right)$ of b_u's $\mathcal{H}$ set with $\ell_u = \ell'_i$.

Note that in addition to b_u's transactions, the miner creates the *coinbase* transaction that awards him for his block creation work. This is the only transaction of b_u whose prefix does not necessarily match label ℓ_u.

Property 3. *The number of leaf blocks self-adapts to transactions demand.*

Property 3 is implemented by having any miner u locally compute the average load of the last $c_{\min}$ blocks of each chain of $\mathcal{L}_u$ to determine the presence of splittable or mergeable leaf blocks. Any splittable block (which is ascribable to a pick of transactions demand, see Definition 2), can only be the predecessor of two split blocks (more precisely of the first blocks of two split chains), each one in charge of its own partition of transactions. Similarly in presence of a drop of transactions demand (see Definition 2), the successor of two sibling mergeable leaf blocks can only be a block in charge of both partitions of transactions. The fact that a block is splittable or two blocks are mergeable, is observable and verifiable by anyone once these blocks are appended to the ledger.

Let us illustrate the construction of block b_u based on the local view $\mathcal{L}_u$ of the ledger at miner u depicted on Figure 1. Miner u progressively computes $\mathcal{H}$ by iterating on all the leaf blocks a, b, c and d of $\mathcal{L}_u$. We have $\mathcal{H} = \left\{\left(\mathfrak{h}(a^{00}), 00, m^{00}\right), \left(\mathfrak{h}(a^{01}), 010, m^{010}\right), \left(\mathfrak{h}(a^{01}), 011, m^{011}\right), \left(\mathfrak{h}(a^{10}), 1, m^1\right), \left(\mathfrak{h}(a^{11}), 1, m^1\right)\right\}$, where $m^{00}, \ldots, m^1$ are the Merkle roots of the four transactions partitions. Notice the presence of the splittable block b and the two mergeable sibling blocks c and d. Then miner u iterates on $\mathfrak{h}(\mathcal{H}||\nu)$ until finding a nonce ν such that $\mathfrak{h}(\mathcal{H}||\nu) \leq T$ with T the current traget of Sycomore. Suppose that the last three bits of $\mathfrak{h}(\mathcal{H}||\nu)$ are equal to 101 (three is the maximal length of the blocks labels that will be appended to the actual leaf blocks a,…,d), then b_u's predecessor are the mergeable blocks c and d, and b_u's label is equal to 1. Finally, miner u includes in b_u's body pending transactions of the partition 1.

In Bitcoin, the rate at which blocks are found is regulated by the difficulty parameter D. The difficulty is simply the ratio between the maximal target and the current target T. It is easier to speak in terms of difficulty than in terms of the target. The maximal target is $(2^{16}-1)2^{208}$ or approximately 2^{224}. A random hash has a chance of about 2^{-32} to be lower than the maximal target. It follows that if the difficulty is D, it takes on average $2^{32}D$ hashes to find a block, and thus assuming the network hashrate is H, then the block creation rate can be approximated as $H/(2^{32}D)$. Every $H_{\max} = 2016$ blocks, the difficulty D is adjusted based on the time it took to mine the last 2016 blocks. In Sycomore, the difficulty is adjusted based on both the network hashrate H and on the number $c \geq 1$ of chains in $\mathcal{L}_u$ (which is an indicator of the current transactions demand). Specifically, adjustment occurs every time all the chains of $\mathcal{L}_u$ have been extended with $H_{\max}$ additional blocks with respect to the last time the difficulty has been adjusted, that is, when their height h satisfies $h = 0 \bmod H_{\max}$.

To cope with the fact that all the chains do not grow at the same rate, and thus do not reach height h at the same instant, once a chain of $\mathcal{L}_u$ has reached height h, then miner u does not take this chain into account to determine the predecessor of its block b_u, i.e., he only considers all the leaf blocks whose height have not reached the re-adjusting point yet. Note that this does not endanger the security of the algorithm as this is verifiable by anyone. By doing so, the gap between the tallest chain and the smallest one rapidly decreases until all the chains reach height h, instant at which the difficulty can be recomputed. A study of this gap is presented in Section IV.

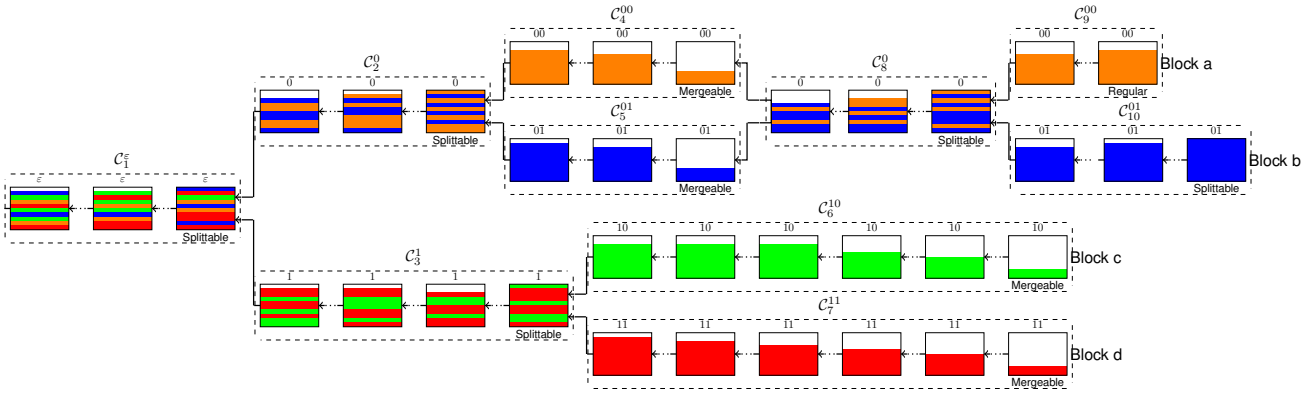

Fig. 1. A possible ledger built with π_{syc}

C. Upon receipt of a block

Once created, blocks are propagated in the system, so that each node v of the system updates its local view $\mathcal{L}_v$ of the ledger $\mathcal{L}$ with the new received block assuming that the block is valid. Block b is valid if

1) the proof-of-work of b is correct, and
2) for each block b' referenced in set $\mathcal{H}$ of b's header, b' belongs to the local view $\mathcal{L}_v$, and
3) all non-coinbase transactions embedded in b match the label of b and do not redeem an already spent output.

If case 2) does not hold (node v may not have received b' yet), b's label cannot be computed and thus case 3) cannot be evaluated yet. Thus b is temporarily stored in v's orphan set, and its validity is postponed until case 2) holds. Note that if case 1) or case 3) is violated, block b is invalid and rejected.

The distributed block creation process may lead to forks, that is the presence of two or more blocks with the same label and pointing back to the same predecessor in $\mathcal{L}_v$ which violates Property 3 of Definition 6. Notice the difference with split blocks. Two split blocks have the same predecessor but each one inherits the label of their predecessor extended respectively with "0" and "1". Split blocks are created as a response to an increase of transactions demand, while forks are due to concurrency. Since the presence of forks gives rise to several concurrent but valid SYC-DAG (see Definition 6), we introduce a rule, Rule 1, that allows any node v to deterministically extract from its local view $\mathcal{L}_v$ a single valid SYC-DAG, denoted by $\mathcal{L}_v^\star$, that respects Property 3 of Definition 6. Note that Figure 1 does not contain any fork, and thus represents a valid SYC-DAG.

Rule 1. *[Fork Rule] At any time, keep the* SYC-DAG *for which the confirmation level of the genesis block is the largest.*

The confirmation level of some block b in $\mathcal{L}_v$ is computed by determining the longest path of blocks that commit the presence of b in $\mathcal{L}_v$. Specifically let F_b rooted at b be the directed acyclic graph that contains all the blocks in $\mathcal{L}_v$ that commit the presence of b in $\mathcal{L}_v$. $F_b = (V_b, E_b)$ where $V_b \subseteq V$ and, for any two blocks $(b_j, b_i) \in V^2$, we have $(b_j, b_i) \in E_b$ if $(\mathfrak{h}(b_i), -, -)$ belongs to $\mathcal{H}_{b_j}$, with $\mathcal{H}_{b_j}$ the header of block b_j as defined in Equation (4). The set V_b is defined by $V_b = \{b_j \in V \mid b_j \rightsquigarrow b\} \cup \{b\}$. Recall that relation $\rightsquigarrow$ refers to the existence of a path from b_j to b. The confirmation level of b is equal to the length of the longest path of F_b. Note that by definition of F_b, this path necessarily ends at b. Surprisingly enough, the fork rule is exactly the same one as in Bitcoin when the SYC-DAG is reduced to a single chain. This makes sense since it amounts to favor the SYC-DAG that has been acknowledged by the largest proportion of miners. Note that the largest confirmation level of the genesis block can be temporarily reached in two concurrent SYC-DAG. By convention, $\mathcal{L}_v^\star$ is kept as long as it is not superseded.

Similarly to Bitcoin, Rule 1 may give rise to the pruning of a chain of blocks due to the presence of a fork. However, once a block has been inserted deep enough in the SYC-DAG then, by construction of the blocks and by Rule 1, with high probability such a block will remain forever in $\mathcal{L}_v^\star$ for any node v. The notion of "deep enough" relates to the confirmation level of a block.

Lemma 1 (Deep confirmation level)**.** *For any node v maintaining $\mathcal{L}_v$, the probability that a block b with confirmation level k is removed from $\mathcal{L}_v^\star$ decreases exponentially with k.*

Proof: Let $b \in \mathcal{L}_v$. By Rule 1, and by the computation of F_b, we can apply the same argument as the one used by Nakamoto [11] and Rosenfeld [14]) to show that the probability that block b remains in $\mathcal{L}_v^\star$ increases exponentially with the length of the longest path of F_b. ■

IV. Properties of the Sycomore protocol

This section is devoted to the study of the properties of the Sycomore protocol, and in particular of $\mathcal{L}_v^\star$, for any node v. We show that at any time a single coherent history exists, meaning that at any time a unique SYC-DAG is the valid SYC-DAG. This is achieved by first showing that $\mathcal{L}_v^\star$ is a prefix of $\mathcal{L}_v$ and that $\mathcal{L}_v^\star$ is a SYC-DAG. We conjecture that for any two correct nodes u and v, $\mathcal{L}_u^\star \sqsubseteq \mathcal{L}_v^\star$ or $\mathcal{L}_v^\star \sqsubseteq \mathcal{L}_u^\star$. Then we study the probability of fork, the confirmation level of transactions, and finally the maximal gap between any two chains at the instant at which the difficulty is readjusted.

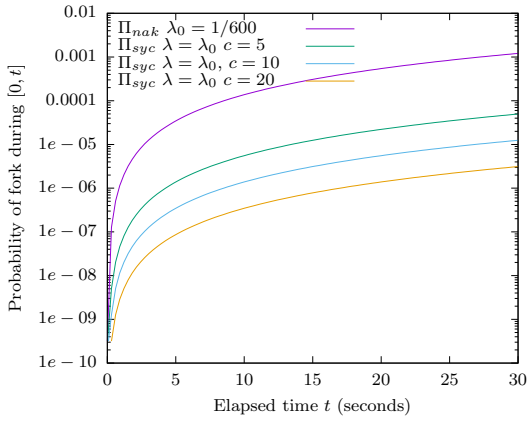

Fig. 2. Probability of fork as a function of time (seconds). The block creation rate is Bitcoin's one, i.e., $\lambda = 1/600$ (one block every 10 mns in average).

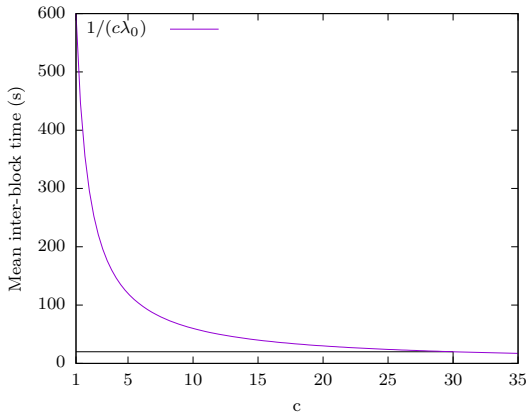

Fig. 3. Mean inter-block time as a function of the number of leaf blocks c to meet Bitcoin's probability of fork (i.e., $p(t) = 0.0012$ which is obtained with $\lambda_0 = 1/600$)

Lemma 2. *For any node v maintaining its the local view $\mathcal{L}_v$, we have $\mathcal{L}_v^\star \sqsubseteq \mathcal{L}_v$*

Lemma 3. *For any correct node v, $\mathcal{L}_v^\star$ is a maximal* SYC-DAG

For space reasons proofs of both lemmata are omitted.

Conjecture 1. *For any two correct nodes u and v, $\mathcal{L}_u^\star \sqsubseteq \mathcal{L}_v^\star$ or $\mathcal{L}_v^\star \sqsubseteq \mathcal{L}_u^\star$.*

Let $\mathcal{L}_v^\star$ be v's SYC-DAG. Suppose that $\mathcal{L}_v^\star$ has c leaf blocks, $b^{\ell_1}, \ldots, b^{\ell_c}$. Let p_i be the probability with which the block creation procedure (see Section III-B) chooses b^{ℓ_i} as predecessor of a new block. We have $\sum_{i=1}^{c} p_i = 1$.

Lemma 4 (Forks occur with low probability)**.** *The probability that two blocks have block b^{ℓ_i} as predecessor during an interval of time $[0, t]$ is $p_i(t) = 1 - e^{-\lambda t/c}(1 + \lambda t/c)$, where λ is the block creation rate.*

Proof: We first suppose that $c = 1$. We model the block creation procedure as a Poisson process. An event represents the creation of a block. Let $\{N(t),\ t \geq 0\}$ be a Poisson process with rate λ representing the number of events in the interval $(0, t)$. We then have, for every $n \geq 0$,

$$\mathrm{P}\{N(t) = n\} = e^{-\lambda t}\frac{(\lambda t)^n}{n!}.$$

For all $t > 0$, we denote by $p(t)$ the probability that at least two events of this process occur in an interval of length t. We then have

$$\begin{aligned} p(t) &= \mathrm{P}\{N(t) \geq 2\} = 1 - \mathrm{P}\{N(t) = 0\} - \mathrm{P}\{N(t) = 1\} \\ &= 1 - e^{-\lambda t}(1 + \lambda t). \end{aligned}$$

We now suppose that $c > 1$. Let $b^{\ell_1}, \ldots, b^{\ell_c}$ be the c leaf blocks of $\mathcal{L}_v^\star$. An event of type i represents the creation of a block that has b^{ℓ_i} as predecessor. The events produced by the Poisson process can be of c different types. We suppose that each event produced is of type i with probability p_i, for $i = 1, \ldots, c$. The successive choices for the types are supposed to be independent of each other and also independent of the Poisson process. For every $i = 1, \ldots, c$, we denote by $\{N_i(t),\ t \geq 0\}$ the number of events of type i produced by the Poisson process. It is well-known that $\{N_i(t),\ t \geq 0\}$ is a also Poisson process with rate λp_i and that these c Poisson processes are independent. We denote by $p_i(t)$ the probability that at least two events of type i occur in the interval $(0, t)$. We then have, for every $i = 1, \ldots, c$,

$$p_i(t) = \mathrm{P}\{N_i(t) \geq 2\} = 1 - e^{-\lambda p_i t}(1 + \lambda p_i t).$$

If $p_i = 1/c$, we get

$$p_i(t) = \mathrm{P}\{N_i(t) \geq 2\} = 1 - e^{-\lambda t/c}(1 + \lambda t/c).$$

This completes the proof of the lemma. ■

Note that case $c = 1$ corresponds to the fork probability in Bitcoin. Figure 2 shows the probability of fork when blocks are created every 10 mn in average (i.e. $\lambda_0 = 1/600$ in Bitcoin's setup). The interval of time that we consider (i.e. 30 seconds) corresponds to the time it takes for a block to be received by a large proportion of the nodes in the peer-to-peer system [5]. This graph shows that the probability of fork in Bitcoin is small (i.e. equal to $p(t) = 0.0012$ in the time interval of 30 seconds) which was confirmed by Decker's observations [5], while in Sycomore this probability of fork is even smaller and decreases as a function of the number of leaf blocks c. For instance the probability that in the time interval of 30 seconds a fork occurs is equal to $p(30) = 4.9 \times 10^{-5}$ when $c = 5$, $p(30) = 1.2 \times 10^{-5}$ when $c = 10$ and $p(30) = 3.1 \times 10^{-6}$ when $c = 20$. This clearly means the capability of Sycomore to sustain a higher rate at which blocks are mined without exceeding Bitcoin's probability of fork. This is shown in Figure 3. In this figure, the relationship between c and the rate at which block are mined to guarantee a probability of fork equal to the one of Bitcoin is depicted. For instance, in presence of $c = 30$ leaf blocks, blocks can be mined every 20 seconds in average while guaranteeing a probability of fork not greater than the one of Bitcoin. This adaptiveness is a remarkable feature of Sycomore, which to the best of our knowledge, is not present in any other ledger design.

TABLE I
VALUES OF t^*, t_1, t_2 FOR DIFFERENT VALUES OF $H_{\max}$ AND a^*.

$(H_{\max}, a^*)$	a	t_1	t^*	t_2
(2016, 1732)	1732	18700	18722	18743
	1500	16225	17516	18811
(10080, 9431)	9431	97507	97537	97567
	7500	77704	87583	97737
(20160, 19237)	19237	196911	196967	197023
	15000	153812	175166	153812

Finally, we focus on the maximal gap that exists between any two leaf blocks at the time the difficulty must be readjusted, that is when the first leaf block reaches height $h = 0 \bmod H_{\max}$. We model the update of the ledger as a ball and urn problem where each produced event is of type i with probability p_i, for $i = 1, \ldots, c$, with c the number of leaf blocks of the ledger. If the events of type i are placed in an urn numbered i, then $N_i(t)$ can be seen as the number of events in urn i at time t. We denote by $M(t)$ and by $m(t)$ respectively the maximun and minimun levels among the c urns at time t. We thus have

$$M(t) = \max_{i=1,\ldots,c}\{N_i(t)\} \text{ and } m(t) = \min_{i=1,\ldots,c}\{N_i(t)\}.$$

We would like to evaluate the instants t at which the level of all the c urns is between a and $H_{\max}$. More precisely we want to determine, for every integer a, $H_{\max}$ and for every $\varepsilon \in (0,1)$, the instants t such that $\mathrm{P}\{M(t) \leq H_{\max}, m(t) \geq a\} \geq 1 - \varepsilon$. For all $t \geq 0$, $H_{\max} \geq 0$ and $0 \leq a \leq H_{\max}$, we have when $p_i = 1/c$,

$$\mathrm{P}\{M(t) \leq H_{\max}, m(t) \geq a\} = \prod_{i=1}^{c} \mathrm{P}\{a \leq N_i(t) \leq H_{\max}\}$$

$$= \prod_{i=1}^{c}\left[\sum_{j=a}^{H_{\max}} e^{-\lambda p_i t}\frac{(\lambda p_i t)^j}{j!}\right] = \left[\sum_{j=a}^{H_{\max}} e^{-\lambda t/c}\frac{(\lambda t/c)^j}{j!}\right]^c.$$

We denote by f this function when $p_i = 1/c$. Then we can show that f has a unique maximum at point $t = t^*$ given by

$$t^* = \frac{c}{\lambda}\left(\frac{H_{\max}!}{(a-1)!}\right)^{1/(H_{\max}-a+1)}.$$

Indeed, introducing the notation

$$v_j(t) = e^{-\lambda t/c}\frac{(\lambda t/c)^j}{j!} \text{ and } u_r(t) = \sum_{j=0}^{r} e^{-\lambda t/c}\frac{(\lambda t/c)^j}{j!},$$

we have $f(t) = \left(u_{H_{\max}(t)} - u_{a-1}(t)\right)^c$ and since $u_r'(t) = -\lambda v_r(t)/c$, then derivative of f is given by

$$f'(t) = c\left(u_{H_{\max}(t)} - u_{a-1}(t)\right)^{c-1}\left(-\frac{\lambda v_{H_{\max}(t)}}{c} + \frac{\lambda v_{a-1}(t)}{c}\right).$$

We then have

$$f'(t) = 0 \Leftrightarrow v_{H_{\max}(t)} = v_{a-1}(t) \Leftrightarrow t = \frac{c}{\lambda}\left(\frac{H_{\max}!}{(a-1)!}\right)^{\frac{1}{H_{\max}-a+1}}.$$

Since $f(0) = 0$ and $\lim_{t\longrightarrow\infty} f(t) = 0$, if $f(t^*) \geq 1 - \varepsilon$ then there exist two values of t called t_1 and t_2 such that for all $t \in [t_1, t_2]$ we gave $f(t) \geq 1 - \varepsilon$.

Numerical application. To illustrate these results, we fix $\varepsilon = 0.01$ and for different values of $H_{\max}$, we compute the greatest value a^* of a such that $f(t^*) \geq 1 - \varepsilon$. We also give for some values of a and $H_{\max}$ the intervals of time $[t_1, t_2]$ in which we have $f(t) \geq 1 - \varepsilon$. We obtain the results shown in Table I where the time is expressed in minutes.

V. CONCLUSION

In this paper, we have proposed Sycomore, a dedicated balanced directed acyclic graph of blocks. Sycomore aims at addressing performance issues of permissionless ledgers. We show that Sycomore allows us to keep all the remarkable properties of the Bitcoin blockchain in terms of security, immutability, and transparency, while enjoying higher throughput and self-adaptivity to transactions demand. We are currently investigating a formal evaluation of Sycomore behavior while working on implementation to bring this solution to a real world usage.

REFERENCES

[1] E. Anceaume, T. Lajoie-Mazenc, R. Ludinard, and B. Sericola. Safety Analysis of Bitcoin Improvement Proposals. In *15th IEEE International Symposium on Network Computing and Applications (NCA)*, 2016.

[2] L. Baird. The SWIRLDS Hashgraph Consensus Algorithm: Fair, Fast, Byzantine Fault Tolerance. http://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf, 2016.

[3] A. Churyumov. ByteBall : A Decentralized System for Storage and Transfer of Value. https://byteball.org/Byteball.pdf, 2017.

[4] C. Decker, J. Seidel, and R. Wattenhofer. Bitcoin Meets Strong Consistency. In *Proc. of the ICDCN International Conference*, 2016.

[5] C. Decker and R. Wattenhofer. Information propagation in the bitcoin in the bitcoin network. In *Procs of the IEEE International Conference on Peer-to-Peer Systems*, 2013.

[6] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the Presence of Partial Synchrony. *J. ACM*.

[7] I. Eyal, A. E. Gencer, E. Gün Sirer, and R. Van Renesse. Bitcoin-NG: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'16, 2016.

[8] J. A. Garay, A. Kiayias, and N. Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques - Advances in Cryptology (EUROCRYPT)*, 2015.

[9] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP, 2017.

[10] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *Proc. of the USENIX Security Symposium*, 2016.

[11] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[12] J. Poon and T. Dryja. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. https://lightning.network/lightning-network-paper.pdf, 2016.

[13] S. Popov. The Tangle. https://iota.org/IOTA_Whitepaper.pdf, 2017.

[14] M. Rosenfeld. Analysis of hashrate-based double spending. http://arxiv.org/abs/1402.2009, 2014.

[15] Y. Sompolinsky, Y. Lewenberg, and A. Zohar. SPECTRE: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive*, 2016, 2016.

[16] Y. Sompolinsky and A. Zohar. Accelerating Bitcoin's Transaction Processing. Fast Money Grows on Trees, Not Chains. *IACR Cryptology ePrint Archive*, 2013, 2013.