An agent-based simulation study of Sycomore ++ , a scalable and self-adapting graph-based permissionless distributed ledger

Emmanuelle Anceaume, Aimen Djari, Sara Tucci-Piergiovanni

An agent-based simulation study of Sycomore^{++}, a scalable and self-adapting graph-based permissionless distributed ledger

Aimen Djari
University Paris-Saclay, CEA, List
Palaiseau, France
mohamed-aimen.djari@cea.fr

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

Sara Tucci-Piergiovanni
University Paris-Saclay, CEA, List
Palaiseau, France
sara.tucci@cea.fr

***Abstract*—The arrival of Bitcoin [1] drove the shift to decentralized ecosystems through the exchange of transactions without intermediary. However, one of the main challenges that need to face permissionless blockchains are scalability and security. In this paper, we present a performance evaluation of Sycomore^{++}, a permissionless graph-based distributed ledger whose main feature is to dynamically self-adapt the number of created blocks to the current number of submitted transactions, and compare them with the ones of Bitcoin and Sycomore, a graph-based distributed ledger. Our evaluation relies on agent-based simulations to evaluate the capability of these distributed ledgers to address the aforementioned challenges, within different execution contexts. One of the main lessons drawn from these intensive simulations is the capability of Sycomore^{++} to drastically reduce transaction confirmation time with respect to the other two ledgers, to quickly react to any sudden variation of the transaction submission rate, to minimize the computational power waste w.r.t. PoW-based permissionless distributed ledgers, and to surpass Bitcoin in terms of resilience to a network adversarial environment. We also study resilience to adversarial miners that want to endanger the quality of the graph showing that, as in Bitcoin, the adversary is limited by its computational power.**

***Index Terms*—Graph-based distributed ledger, scalability, distributed ledger quality, agent-based simulation**

I. Introduction

A recent evolution in blockchain technology seeks to address the performance issue of permissionless chain-based ledgers, in particular the small number of transactions confirmed per second – around $7tx/s$ for Bitcoin. While some new efforts are dedicated to replace the proof-of-work (PoW) consensus mechanisms with mechanisms such as proof-of-stake and BFT consensus (such as [2]–[5]), reaching $10^2 - 10^3$ tx/s, it is undeniable that Bitcoin has shown a great longevity, validating on the ground its good design and security properties in these last 10 years. For this reason other proposals are exploring how to leverage the same design principles, and in particular the simplicity, of Bitcoin protocol. In this line of works some proposals, including [6]–[8], called second-layer protocols, propose to implement a protocol on top of Bitcoin that hits the blockchain only from time to time. In this way second-layer transactions are handled at the Internet speed, while only special transactions, needed occasionally to open/close sessions and solve disputes, are translated into Bitcoin transactions. While the idea of off-loading transactions is interesting, these proposals do not specifically address the problem of scalability of the ledger-based PoW itself. In this respect, and to the best of our knowledge (see Section II), Sycomore [9] [1] has been the first ledger protocol, relying on Bitcoin design principles, that addresses Bitcoin's scalability issues: Its graph-structure design allows for the "parallel" creation of valid and durably appended chains of blocks. The unique feature of Sycomore is that its graph structure dynamically adapts to fluctuations in transaction submission rates: when the last blocks appended to a given chain $\mathcal{C}$ of the graph exceed some maximal loading threshold, subsequent blocks of transactions are partitioned over two created sibling chains referencing $\mathcal{C}$, and these blocks are mined in parallel. Conversely, when the last blocks of two sibling chains $\mathcal{C}_i$ and $\mathcal{C}_j$ fall short of a minimal loading threshold, subsequent blocks will belong to a unique chain, referencing both $\mathcal{C}_1$ and $\mathcal{C}_2$. The decisions to split a chain of the graph or to merge two sibling ones is locally taken by each miner, and soundness of this decision is verifiable by everyone at any time [9].

A. Motivations of this Work.

By dynamically adapting the width of the graph to the actual transaction load of the network, one might expect that Sycomore would guarantee an almost optimal transaction latency. By transaction latency it is meant the time that elapses between the instant at which a user submits a transaction to the network and the instant at which this transaction is confirmed, i.e., belongs to a block that is deeply settled down into one of the chains of the graph. Transaction latency deeply depends on the mining difficulty, i.e. the difficulty to create a block. To cope with variations of the network computational power, the mining difficulty is periodically readjusted, to guaranteeing both security and acceptable latency. In Bitcoin, such a readjustment is executed every time the height of the blockchain has been increased by 2016 blocks, i.e. every 14

[1] Sycomore is the french word for sycamore, a large broad-leaved tree tolerant to wind.

days. Sycomore has a similar readjusting scheme, proceeding to a readjustment of the difficulty every time the height of the graph has been increased by $H_{max} = 2016$ blocks since the previous readjustment. Since Sycomore can have more than one chains in the graph, the mining difficulty is adjusted to fit the width of the graph, i.e., the number of leaf chains in the graph. Unfortunately, the number of leaf chains between any two readjustments of the difficulty can dynamically vary to cope with variations of transaction submission rates. Hence, the mining difficulty which was computed to fit both the hash rate of the network and number of chains at the last readjustment may currently be either under-estimated or over-estimated. To illustrate this point, let us consider the scenario where, at the time readjustment took place, the graph's width was very large, but subsequently the number of submitted transactions significantly dropped, leading to a progressive diminution of the number of leaf chains, and thus an under-estimated mining difficulty. Such a scenario shows a possible breach of security: adversarial miners can take advantage of a too low mining difficulty to degrade the ledger *quality* [10], that is the maximal proportion of blocks contributed by the adversary in a sufficiently long part of the ledger maintained by a honest node. The opposite scenario may also happen, where a sudden augmentation of the transaction rate will give rise to an over-estimated mining difficulty, and thus a very high power consumption, which will be in the worst case similar to Bitcoin's one. As a consequence, transaction latency will be very high, until the next readjustment of the difficult takes place, thus hindering scalability.

B. Contributions of this work

This work presents a twofold contribution. First we present Sycomore^{++}, a truly scalable proof-of-work based protocol that solves the critical issues mentioned above. Sycomore^{++} inherits the main mechanisms of Sycomore, while adding a new mechanism to adjust difficulty in such a way that at any time a constant inter-block creation delay is maintained on any leaf chains of the graph. Secondly, we propose fine-grained simulations to evaluate and compare protocols that have dynamic behavior over complex graph structures. More in detail, to finely validate and compare the behavior of Sycomore^{++} with respect to its direct competitors, we have implemented Bitcoin, Sycomore and Sycomore^{++} on an agent-based simulator and have compared these three protocols in presence of adversarial environments, i.e., sophisticated attacks, large communication delays, and sudden and substantial variations of the system workload demand. Lessons learnt from these experiments show that Sycomore^{++} succeeds in providing (*i*) a transaction confirmation rate varying linearly with the transaction submission rate, a very small and almost optimal average transaction latency regardless of the submission transaction rate, a negligible number of transactions pending at miners prior to be embedded in blocks, a drastic reduction of the computational power used to create blocks that will never appear in the ledger w.r.t Bitcoin and Sycomore. Finally, we have designed sophisticated attacks that an adversary may elaborate to hinder Sycomore^{++}'s quality property [10]. Garay et al. [10] define the ledger quality as the property that ensures that an adversary cannot control more than a $\mu/(1-\mu)$ percentage of the blocks in a sufficiently long part of the ledger, where μ represents the ratio of the network hashing power owned by the adversary. In this respect, and to the best of our knowledge, this paper presents one of the most advanced studies of fine-grained agent behaviors in graph-based ledgers.

The remaining of the paper is organized as follows. Section II discusses related work on simulation and graph-based distributed ledgers. To make the paper self-contained, a brief overview of Sycomore is presented in Section III, while Section IV presents Sycomore^{++} and an analytical performance study. Section V highlights the main lessons learnt from the experiments we have conducted to validate Sycomore^{++} and to compare its performance with both Bitcoin and Sycomore. Finally, Section VI concludes the paper.

II. Related Work

Modeling and Simulation. To shed some light on the dynamics of blockchain systems, recent research focused on suitable simulation models to capture their behavior. The different proposals differ in the level of abstraction of the model, the simulation type and the properties under study. Kaligotla et al. [11] propose an agent-based framework for evaluating distributed ledgers, modelling a blockchain at a very abstract level as a simple append-only queue where a block is added when verified by enough agents. Agents issue transactions and verify them through atomic actions (no messages are exchanged) with associated costs, fees and energy costs, respectively. In the present work we consider a simulation model at a finer granularity level, in the sense that the proof-of-work is simulated for each agent according to its computational power, and a large range of transaction submission rates, network delays, and adversarial strategies are considered. Piriou et al. [12] propose a stochastic simulation model and Monte-Carlo simulations to evaluate the performance of a blockchain when the communication system loses messages. Double-spending attacks are modeled by a simplified append-only queue model (which considers instantaneous communication). Alharby et. al. [13], Rosa et al. [14] and Faria et al. [15] propose discrete-event simulators for distributed ledgers that resemble to Bitcoin and Ethereum, but not adapted to simulate graph-based distributed ledgers. On the other hand, Bottone et al. [16] present a simulation model for Tangle-like DAG ledgers [17] aiming at studying the grow of the graph of transactions. The simulation model is instrumented on the NetLogo [18] agent-based simulator. Due to the complexity of the computational model, simulated strategies are very simple and network effects are not taken into account.

Graph-based protocols. Many graph-based protocols have been explored in the last years. Proposals such as HashGraph [19], ByteBall [20], and Iota [17], [21] do not use blocks, i.e. a graph is formed by transactions pointing

to each other. However, these solutions are not fully decentralised because they leverage the presence of central or trusted nodes. Ghost [22] and Spectre [23] protocols keep blocks, modifying the blockchain data structure from a totally ordered sequence of blocks to a directed graph of blocks. Note that in these approaches, the absence of mechanisms to prevent the presence of conflicting records (i.e., blocks with conflicting transactions) or the presence of cycles in the directed graph (Spectre [23] organises blocks in a directed, but not acyclic, graph of blocks) require that participants execute a complex algorithm to extract from the graph the set of accepted (i.e., valid) transactions [23]. Sycomore has been the first graph-based protocol to be fully distributed. In contrast to previous approaches, neither a chain of blocks nor a set of transactions are extracted from the graph to become the valid blockchain or the valid set of transactions. Instead, the full graph is the ledger. Blocks are built so that they commit the state of the directed graph at the time blocks were created, which decreases the opportunity for powerful attackers to create blocks in advance.

III. Overview of Sycomore

Sycomore [9] is a cryptocurrency ledger whose structure is a dedicated balanced directed acyclic graph of blocks called the SYC-DAG. Construction of the SYC-DAG is very close to Bitcoin one, i.e., it is fully distributed, permissionless, and relies on a proof-of-work mechanism. Sycomore enjoys a set of properties that enable it to dynamically adapt the fan-out of the SYC-DAG to the current number of transactions submitted to the system.

A. Properties of Sycomore

Sycomore has been designed to meet the following properties [9]:

- **P1. Self-adaptation to transaction load.** A rise or a drop in the current number of submitted transactions is dynamically handled by the progressive creation or disappearance of sibling leaf chains in the SYC-DAG;
- **P2. Balanced partitioning of transactions.** There does not exist any transaction that belongs to two different blocks.
- **P3. Unpredictability of the predecessor.** The leaf chain to which a new block is appended can neither be chosen nor predicted among all the leaf blocks of the SYC-DAG.
- **P4. Chain fairness.** All the leaf chains of the SYC-DAG grow at the same speed.
- **P5. Negligible probability of forks.** The probability of forks is maximal when the SYC-DAG is reduced to a single chain (i.e, $1,2 \times 10^{-3}$ in the time interval of 30 seconds) and decreases proportionally with the number of leaf blocks.

To make the paper self-contained, we detail in this section how Sycomore implements those five properties.

Property P1 is implemented by introducing the notion of *splittable* and *mergeable* blocks [9], which are a dynamic response to respectively a rise or a drop in the current submission rate of transactions in the system. Both notions refer to block load, where the load of a block is the ratio between its number of bytes and its maximal load (for instance, 1 MByte in Bitcoin prior to the date of SegWit activation). Hence, a block b appended to the SYC-DAG is called *splittable* if the average load of block b together with the load of its $c_{\min}-1$ predecessors on the chain exceeds the overload threshold Γ (both $c_{\min}$ and Γ are system parameters). When a block b is splittable, miners will create subsequent blocks so that they will form two parallel chains of blocks, such that the first block of each of both chains references b. These chains are called sibling chains. Partitioning of the transactions over the chain blocks is explained when Property P2 is explained. Conversely, when the transaction rate drastically drops, the block load decreases accordingly, leading Sycomore to progressively reduce the number of chains in the SYC-DAG to keep blocks sufficiently loaded. Specifically, a block is called *mergeable* if the average load of this block together with the load of its $c_{\min}-1$ successive predecessors of its chain falls short of some given underload threshold γ (γ is a system parameter). When two blocks belonging to two sibling chains are mergeable, miners will create subsequent blocks so that they will form a single chain (called merged chain). Any block that is neither mergeable nor splittable is said *regular*. As argued in [9], it is clear that everyone, and in particular miners, detect the instant at which a block is splittable or two sibling blocks are mergeable. This is observable and verifiable by anyone since it only depends on a publicly observable quantity (i.e., block load).

Property P2 aims at fully exploiting the gain brought by sibling chains, i.e, the effective partitioning (in the mathematical sense) of the transactions over the SYC-DAG. This property is implemented by introducing the notion of *label*. A label is a binary string, and characterizes the common prefix of the identifier (i.e. fingerprint) of the set of transactions embedded in a block. Any block when created is tagged with the label of the chain it will belong to (the choice of the chain a block will belong to is explained when Property P3 is discussed). The genesis block b_0 is labelled with the empty binary string ε, and all the blocks from b_0 to the first splittable block b (if any) of the chain, say $\mathcal{C}_1^{\varepsilon}$ in Figure 1, are labelled with the empty string ε. Hence, all the blocks of $\mathcal{C}^{\varepsilon}$ contain transactions for which there is no constraint on the prefix of their identifier (this reflects Bitcoin's behavior). On the other hand, all the blocks of two sibling chains, say $\mathcal{C}_2^0$ and $\mathcal{C}_3^1$ in Figure 1, appended to the splittable block b inherit b's label extended with 0 and 1 respectively. Hence, all the blocks of $\mathcal{C}_2^0$ (resp. $\mathcal{C}_3^1$) only contain transactions whose identifier is prefixed by 0 (resp. 1). As transactions' identifiers can be considered as random bit strings, transactions are evenly partitioned over sibling chains, and transactions cannot appear in more than one block, which makes the parallelism introduced by the graph structure effective. The same process applies for any splittable block. Conversely, all the blocks that belong to a merged chain inherit the largest common prefix of its predecessor labels. For instance, in Figure 1, chains $\mathcal{C}_4^{00}$ and $\mathcal{C}_5^{01}$ give rise to the merged chain $\mathcal{C}_8^0$ whose label is the largest common prefix of

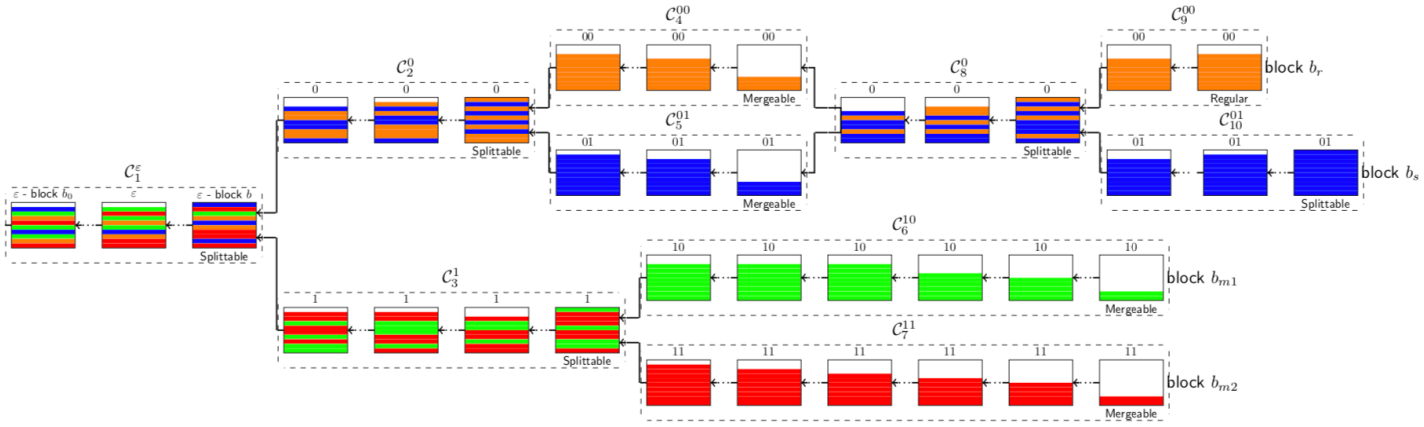


Fig. 1: An example of a SYC-DAG built by Sycomore. This figure has been borrowed from [9]. System parameters: overload threshold $\Gamma = 95\%$, underload threshold $\gamma = 15\%$, and $c_{\min} = 2$. The number of bars in each block is representative of block load, and colors of the bars illustrate the prefix of transaction identifiers. This provides an intuitive way to see when chains split or merge, and how transactions are partitioned over the SYC-DAG: When the SYC-DAG is made of a single chain due to a very light load (e.g., chain $\mathcal{C}_1^{\varepsilon}$), each block contains transactions whose identifiers are prefixed with the empty binary string label (denoted by ε), which explains the multitude of colors of the blocks (which exactly reflects Bitcoin's chain). When the chain must split into two sibling chains because of an increasing transaction load, the new appended blocks partition the transactions into two sets: those whose prefix match label ε concatenated with 0, i.e, label 0, and those whose prefix match label ε concatenated with 1, i.e, label 1. This explains the partitioning of block colors in the upper and lower chains respectively. A similar argument applies when sibling chains merge to a single chain: subsequent blocks of this chain will contain transactions whose identifier is prefixed by the largest common prefix of the labels of these sibling mergeable chains (e.g., label 0 in chain $\mathcal{C}_8^0$). Chains $\mathcal{C}_9^{00}$, $\mathcal{C}_{10}^{01}$, $\mathcal{C}_6^{10}$, and $\mathcal{C}_7^{11}$ are called leaf chains, as blocks b_r, b_s, b_{m1} and b_{m2} are the leaf blocks of the SYC-DAG. Note that this SYC-DAG does not contain any fork.

both labels 00 and 01, i.e., 0.

Property P3 is implemented by using the unpredictability and randomness of the proof of work (PoW) to assign the *predecessor* of any block b. To make such an assignment immutable, verifiable by anyone and non-ambiguous, the header of any block b contains a set of tuples that *(i)* acknowledges or commit the miner's local view of the SYC-DAG, and *(ii)* characterizes b's predecessor. More precisely, let $\mathcal{L}_u$ be the local view of the SYC-DAG at miner u. Suppose that $\mathcal{L}_u$ contains c leaf blocks $b^{\ell_1}, \ldots, b^{\ell_c}$ at the time u starts b's creation process, and among these c leaf blocks, s of them are splittable, [2] miner u builds b's header as follows: it inserts, among different pieces of information, a set of $(c+s)$ commitment tuples

$$\{\ldots, (H(b^{\ell_j}), \ell'_j, m^{\ell'_j}), \ldots\},$$

where, for $1 \leq j \leq c$, $H(b^{\ell_j})$ is a cryptographic link to leaf block b^{ℓ_j}, ℓ'_j is the label of the block for which b^{ℓ_j} will be the predecessor, and $m^{\ell'_j}$ is the Merkle root of the set of locally pending transactions whose identifier is prefixed by ℓ'_j. If leaf block b^{ℓ_j} is splittable then two tuples $(H(b^{\ell_j}), \ell_j 0, m^{\ell_j 0})$ and $(H(b^{\ell_j}), \ell_j 1, m^{\ell_j 1})$ commit the presence of block b^{ℓ_j} in $\mathcal{L}_u$. If leaf blocks $b^{\ell_j 0}$ and $b^{\ell_j 1}$ are both mergeable and belong to sibling chains then two tuples $(H(b^{\ell_j 0}), \ell_j, m^{\ell_j})$ and $(H(b^{\ell_j 1}), \ell_j, m^{\ell_j})$ commit the presence of those mergeable blocks in $\mathcal{L}_u$. By doing this, block b extends $(c+s)$ commitment paths, one to each leaf block of $\mathcal{L}_u$, and recursively down to the genesis block. The length of a commitment path (that is the number of blocks on the path) is used to resolve forks if any (see Rule 2). Miner u then engages in finding a nonce ν such that ν is the solution of the PoW applied on b's header (exactly as in Bitcoin). If successful, the predecessor of block b is the leaf block b^{ℓ_i} in $\mathcal{L}_u$ closest to ν. More precisely, for each tuple $(H(b^{\ell_j}), \ell'_j, m^{\ell'_j})$ in b's header, the numerical value of the "exclusive or" (XOR) between ν and ℓ'_j is computed, and the winning tuple is the one that minimizes this distance. Let $(H(b^{\ell_i}), \ell'_i, m^{\ell'_i})$ be that tuple. The *predecessor* of block b is thus the leaf block b^{ℓ_i}. Miner u completes the creation of its block b by embedding the appropriate set of transactions, that is the set of transactions whose identifier is prefixed by ℓ'_i and whose Merkle root is $m^{\ell'_i}$. Extracting b's predecessor from the PoW computed for b makes the choice of block's predecessor an unpredictable and random process. Notice that no specific reference to b's predecessor is added in b's header: b's header is securely sealed with PoW ν, and thus when a node receives block b, it derives b's predecessor by using the information in b's header (i.e. ν and the set of tuples).

Property P4 follows from the assumption that the PoW is modeled by a random oracle, and that transaction identifiers result from the SHA256 cryptographic hash function.

Property P5 directly derives from Properties P3 and P4: since each created block is appended to a random leaf block, the probability that two blocks with the same label share the same predecessor (this is a fork situation) is equal to p/c, where p is the probability of fork in Bitcoin, and c is the

[2] Note that a splittable block b_s is considered a leaf block as long as b_s is not the predecessor of two sibling blocks.

current number of leaf blocks in the SYC-DAG.

Based on the above descriptions, a SYC-DAG is defined as follows.

Definition 1 (SYC-DAG [9])**.** *A graph $G = (V, E)$ is a SYC-DAG if G has a unique genesis block b_0 and there exists a partition $\mathcal{P} = \{\mathcal{C}^{\ell_1}, \ldots, \mathcal{C}^{\ell_n}\}$ 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:*

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

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

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

Similarly to all PoW-based distributed ledgers, the distributed block creation process may lead to forks, that is the presence of at least two concurrent blocks appended to the ledger. In Sycomore two blocks are concurrent if and only if both blocks have the same label and the same block predecessor (which differs from split situations). The presence of forks gives rise to concurrent SYC-DAGs $\mathcal{L}_u$ and $\mathcal{L}'_u$ (both of them being rooted at the genesis block). To resolve forks, that is to locally keep a single SYC-DAG $\mathcal{L}^\star_u$, i.e, $\mathcal{L}^\star_u = \mathcal{L}_u$ or $\mathcal{L}^\star_u = \mathcal{L}'_u$, node u applies the fork rule described below. This rule relies on the confirmation level of a SYC-DAG. By definition, the confirmation level of a SYC-DAG is equal to the number of blocks that belong to the longest commitment path (as defined earlier in this section) that commit the presence of the genesis block in this SYC-DAG.

Rule 2 (Fork rule [9])**.** *At any time, keep the SYC-DAG $\mathcal{L}^\star$ for which the confirmation level of the genesis block is the largest.*

As for Bitcoin, the fork rule favors the SYC-DAG that has been acknowledged by the largest proportion of miners. Note that two concurrent SYC-DAGs may temporarily have the same confirmation level. By convention, the oldest SYC-DAG is kept as long as it is not superseded. Once a block has been inserted deep enough then by construction of the blocks and by Rule 2, with high probability such a block will remain forever in the local view $\mathcal{L}^\star_u$ of any node u in the system. The notion of "deep enough" relates to the block confirmation level.

B. Periodic readjustment of the difficulty

To guarantee that the creation time between any two successive blocks of any given chain is constant in average, the mining difficulty D is periodically adjusted based on the current network hashrate (which is reflected by the time it took to mine the last blocks of the SYC-DAG) and the current number $c \geq 1$ of leaf blocks. Specifically, adjustment of the difficulty takes place every time the height of the SYC-DAG has been increased by h blocks with respect to the last time the difficulty was adjusted, that is, when its height h satisfies $h = 0 \bmod H_{\text{max}}$, with $H_{\text{max}} = 2016$. To cope with the fact that some of the leaf chains may grow a little bit slower than others, and thus leaf blocks do not reach height h at the same instant, once a leaf chain has reached height h, miners do not take this leaf chain into account to determine the predecessor of their block, i.e., they only consider all the leaf blocks whose height have not reached height h yet. Once all the leaf chains have reached height h, miners readjust the difficulty, if needed. Note that there is no incentive for an adversary not to follow this rule since its new block will be rejected by the other miners.

On the other hand, it is likely that between any two periodic readjustments of the difficulty, the width of the SYC-DAG drastically increases or decreases according to the transaction load demand (a cascade of splits or merges is observed when the submission transaction rate varies as observed in Section V-D). This will lead to an inappropriate difficulty whose impact is twofold: from a security point of view, it may degrade the ledger quality, that is the maximal proportion of blocks contributed by the adversary that belong to the SYC-DAG of any honest node. From a progress point of view, if the difficulty becomes too high, the average creation delay between any two successive blocks will drastically increase, augmenting accordingly transaction latency, and thus transaction confirmation delay.

IV. SYCOMORE^{++}

To prevent such critical issues, we propose Sycomore^{++}, which aims at guaranteeing that whatever the structure of the SYC-DAG, a constant inter-block creation delay is maintained on any of its chains. The main idea of Sycomore^{++} is to continuously adapt the block creation difficulty to the actual number of leaf chains of the SYC-DAG. Note that this adaptative adjustment does not replace the periodic global computational power readjustment. The former adapts the difficulty to the structure of the SYC-DAG while the latter adapts the difficulty to the hashing power of the system.

Lemma 3. *The expected effort miners must exert in Sycomore^{++} to successfully create a block decreases with the number of leaf blocks of the SYC-DAG.*

Proof. Let $\mathcal{U}$ be the current set of miners that participate to the construction of the SYC-DAG. We suppose that the computational power of the network is uniformly distributed among all the miners in $\mathcal{U}$. Producing a proof of work is a random process with low probability of success so that a lot of trials and errors are required on average before a valid proof of work is generated, the probability of success p_{pow} of each trial being the same. The Geometric distribution models the number of failures before the first success. Thus, if random variable X represents the number of failures before the first success, we have $P(X = q) = (1 - p_{pow})^{q-1} p_{pow}$. Let W be the total computational power of the system. In Sycomore^{++}, the difficulty is divided by the current number c of leaf chains in the SYC-DAG (Recall that in Sycomore, this does not necessarily hold, in particular in presence of variations of the system workload demand). This comes backs to multiplying the probability of success p_{pow} of the PoW process by c. The

probability p of successfully mining a block is thus given by $p = W \times p_{pow} \times c$, and at a miner, this probability is equal to $p_u = p/|\mathcal{U}|$. In Sycomore^{++}, the probability for a miner to work on a given chain is $1/c$, and the average number n_c of miners working on a chain is equal to $|\mathcal{U}|/c$. Thus the probability p_c of successfully mining a block on a given leaf chain p_c given by $p_c = p/c$. Let X_c be the random variable representing the number of trials before the first success on a given leaf chain, we have $\mathbb{E}(X_c) = 1/p_c = 1/(W \times p_{pow})$, and consequently, $\mathbb{E}(X) = 1/p = 1/(W \times p_{pow} \times c)$. □

This lemma shows that in Sycomore^{++} the expected number of unsuccessful trials before creating a block decreases with the number of leaf blocks, which is not the case in Sycomore. This demonstrates the exemplary behavior of Sycomore^{++}: Both its SYC-DAG structure and the mining difficulty self-adapt to the current number of transactions submitted to the system, which allows it to operate in adversarial environments in which the transaction load can change arbitrarily. This is confirmed by the experimental evaluation presented in Section V.

The following Lemma shows that the occurrence of forks decreases exponentially with the number of leaf chains.

Lemma 4. *[9] Given a ledger $\mathcal{L}_v^\star$ with c leaf chains $\mathcal{C}_1, \ldots, \mathcal{C}_c$, each one being selected by the block creation process with probability p_i, with $\sum_{i=1}^{c} p_i = 1$, the probability that two blocks extend the very same chain $\mathcal{C}_i, 1 \leq i \leq c$ 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. Let us first consider the case where $c = 1$. We model the block creation process as a Poisson process. In the following an event represents the creation of a block. Let $\{N(t),\ t \geq 0\}$ with rate λ, be the Poisson process representing the number of events in the interval $(0, t)$. We then have, for every $n \geq 0$,

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

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.

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

Let us assume that the SYC-DAG contains $c \geq 1$ leaf blocks, $b_1^{\ell_1}, \ldots, b_c^{\ell_c}$. The probability p_i for a newly created block to have $b_i^{\ell_i}$ as predecessor depends on $b_i^{\ell_i}$'s header. The events produced by the Poisson process can be of c different types. An event of type i represents the creation of a block that matches chain $\mathcal{C}_i^{\ell_i}$. Each event produced is of type i with probability $p_i = 1/c$, 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 $i = 1, \ldots, c$, let $\{N_i(t),\ t \geq 0\}$ be the number of events of type i produced the Poisson process. It is well-known that $\{N_i(t),\ t \geq 0\}$ is a also Poisson process with rate $\lambda \times p_i$ 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 $(0, t)$ (i.e. a fork occurs in the interval $(0, t)$). We then have, for every $i = 1, \ldots, c$,

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

It is interesting to remark that this probability holds in Sycomore only at the instants at which readjustments of the difficulty occur. □

Lemma 5. *For any correct node u, $\mathcal{L}_u^\star$ does not contain double-spending transactions*

Proof. The proof is by contradiction. Suppose that $\mathcal{L}_u^\star$ contains two transactions $T_1 = (I_1, O_1)$ and $T_2 = (I_2, O_2)$ such that T_1 and T_2 redeem a common UTXO o, where o belongs to the output set of some transaction $T \in \mathcal{L}_u^\star$. Suppose that T_1 and T_2 respectively belong to blocks b_1 and b_2. Since b_1 and b_2 belong to $\mathcal{L}_u^\star$ both blocks are valid. Two cases must be considered.

- $b_1 = b_2$. This case is impossible since it would mean that block $b_1 = b_2$ is invalid (i.e., it contains conflicting transactions T_1 and T_2).
- $b_1 \neq b_2$. Suppose without loss of generality that node u already appended b_1 to $\mathcal{L}_u^\star$ by the time it wishes to append b_2. Two sub-cases are possible.
  - b_2's header commits the existence of b_1 in $\mathcal{L}_u^\star$, that is b_2's header extends at least one commitment path that acknowldges the presence of b_1 in one of the chains of $\mathcal{L}_u^\star$. This contradicts the assumption that b_2 is valid.
  - b_1 and b_2 have been concurrently mined, that is at the time both blocks were mined their respective miners did not know the existence of the other block. By assumption, $b_1 \in \mathcal{L}_u^\star$ at the time node u wishes to append b_2. Since the presence of $b_1 \in \mathcal{L}_u^\star$ makes block b_2 invalid, node u will reject block b_2. This contradicts the assumptions that $b_2 \in \mathcal{L}_u^\star$. Note that another node v may have first appended b_2 to its ledger $\mathcal{L}_v^\star$, and thus will reject block b_1. Eventually, either $\mathcal{L}_v^\star$ or $\mathcal{L}_u^\star$ will contain the longest commitment path to the genesis block, and thus by Rule 2, the ledger with the longest commitment path to the genesis block will be kept by all the nodes. This completes the proof.

□

V. Simulation Study

This section presents the agent-based simulation study we have conducted on Bitcoin, Sycomore and Sycomore^{++}. The source codes of Bitcoin, Sycomore and Sycomore^{++} as well as all the scripts of the experiments are publicly accessible [24].

A. Simulator and Experimental Environment

We have used an agent-based simulation framework dedicated to blockchain systems, called Multi-Agent eXperimenter (MAX) [25] based on the MaDKit framework [26]. MAX offers generic libraries to easily develop distributed ledger protocols and a large range of simulation scenarios. The simulator

is a discrete event simulator, where the unit of simulation time is referred to as a tick. Message-passing libraries allow us to configure different types of communication schemes and message delays. In this work, the communication schema is configured as a reliable broadcast with configurable delay. Impact of message losses is left for future works. All the experiments for Sycomore^{++}, Sycomore and Bitcoin have been run on Grid'5000, a large-scale and flexible test-bed for experiment-driven research [27]. Due to the computational complexity of simulation models and experiments involving a representative number of agents, each experiment presented in this paper takes in average 8 hours.

B. Simulation Model

1) Block creation model: Miners create blocks by following the prescribed protocols, i.e., validation of the set of transactions to be inserted and creation of block header. For straightforward reasons, miners do not solve proof-of-works but follow a simulation model. Before disseminating a block to the network, a miner waits for a time determined by the PoW model described below. Note that for both Sycomore^{++} and Sycomore, the selection of the random predecessor in the model is achieved by computing the distance between the block header (rather than the PoW nonce ν, which is not computed in the model) and each leaf block of the SYC-DAG.

2) Proof-of-Work Model: To simulate the effort needed to find the proof, each miner $u \in \mathcal{U}$ waits for a certain amount of ticks, which depends on its computational power W_u. Specifically, W_u is a fraction of the global computational power distributed among miners according to a power law distribution (with parameter 3) such as $\sum_{u \in \mathcal{U}} W_u = 1$. The probability for miner u to solve the proof-of-work after ℓ successive independent draws is modeled as a geometric distribution with parameters ℓ and p_{POW}, where p_{POW} is the probability of successfully solving the Proof-of-Work, i.e., $p_{POW} = D/2^k$, where k the security parameter of the Proof-of-Work and D the difficulty level. Difficulty and security parameters have been set such that for $W = 1$, the time to solve the proof-of-work is 10 ticks in expectation. Calibration of our model has been set by using Bitcoin real network statistics and by running our model with data extracted from the real network using tools presented in [28].

3) Common parameters of the simulations: For all the experiments presented in the paper we have fixed some common parameters as follows:
- The block capacity, that is the maximal number of transactions a block can embed, is set to 100 transactions (to avoid the simulator overload). Note that while in Bitcoin the block capacity is approximately equal to $4,000$ transactions [29], reducing the block capacity does not affect the behaviour of the protocols.
- A transaction is confirmed when the block this transaction belongs to has a confirmation level equal to $k = 6$. Recall that the confirmation level of any block b is equal to the number of blocks that confirm the presence of b in the blockchain. Note that differently from Bitcoin, in both Sycomore and Sycomore^{++}, these blocks can belong to different chains of Sycomore, as long as these blocks form a path of commitment down to block b.
- c_{min} is set to 1. Impact of c_{min} on the structure of the SYC-DAG and its performances is left for future works.
- For each experiment, we have run sufficiently many simulations to get a confidence interval equal to $5 \pm \%$.

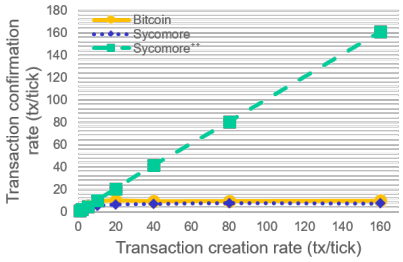
C. Scalability Study

This section studies the capability of Sycomore^{++}, Sycomore and Bitcoin to handle high transaction submission rates. Specifically, we evaluate the transaction confirmation rate, the transaction latency, i.e., the average time elapsed between the submission of a transaction in the network and the time at which the transaction is confirmed, and the average number of pending transactions at the end of the simulation (i.e., waiting to be embedded in a block). The energy lost by each protocol is also measured. The lost energy is the sum for each miner u and for each created block b not appended to the distributed ledger of the time spent working on b times the computational power cp_u. In this section, we assume that forks do not occur.

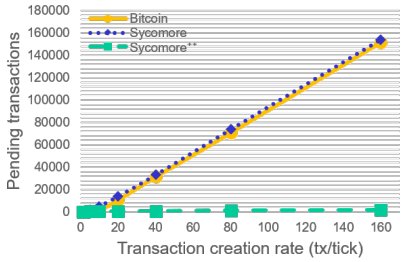
1) Experiment setting: The overload threshold Γ which conditions the SYC-DAG splitting in both Sycomore and Sycomore^{++} varies from 90% to 100%. Note that when $\Gamma = 100\%$, splits never occur and thus both Sycomore and Sycomore^{++} reduce to Bitcoin. The submission rate of transactions f_{req}, which represents the number of transactions submitted per tick of simulation, is set at the beginning of each experiment. f_{req} varies from 1 to 160 txs/tick. Let us remark that we tune the proof-of-work parameters to get in expectation one block mined every 10 ticks. This means that in Bitcoin $f_{\text{req}} = 10$ txs/tick already exhausts the system transaction treatment capacity, as the system mines one block every 10 ticks in expectation and one block contains 100 transactions. From this observation, we might expect that for $f_{\text{req}} > 10$ txs/tick, pending transactions will accumulate over time in, at least, Bitcoin ledger. Note that to avoid the overload of the simulator we were limited to $f_{\text{req}} = 160$ txs/tick. Anyway, setting f_{req} up to 160 txs/tick allows us to severely stress Bitcoin, Sycomore and Sycomore^{++}. Similarly to Bitcoin Core client, miners give priority to old transactions in Bitcoin, Sycomore and Sycomore^{++}.

2) Experiment results: The main results of our experiments appear in the graphs of Figure 2. Note that in all the graphs, points are linked together with lines. This is only for readability reasons.

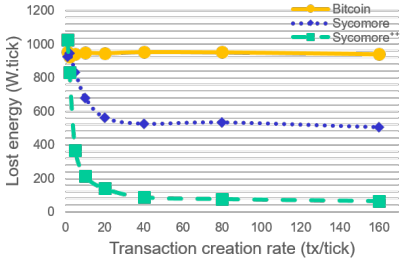
Let us first focus on the confirmation rate of transactions as a function of their creation rate (see Figure 2a). The main observation regarding Bitcoin and Sycomore is that whatever the computational power of the network, no more than 10 txs/tick are confirmed, which illustrates the impact of the globally constant inter-block delay (i.e. a block is mined every 10 ticks in average). Sycomore shows slightly worse results than Bitcoin, which is due to the augmentation of the number of chains, in which blocks can be moderately loaded. On the other hand, by continuously adapting the mining difficulty



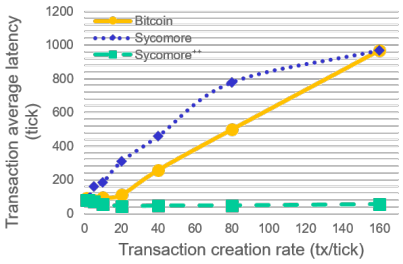
(a) Transaction confirmation rate as a function of the transaction creation rate.



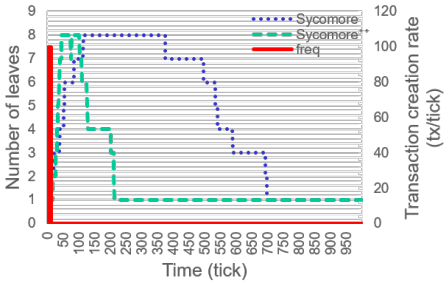
(b) Number of pending transactions as a function of the transaction creation rate.



(c) Lost energy as a function of the transaction creation rate.



(d) Transaction average latency as a function of the transaction creation rate.



(e) Reactivity of both Sycomore and Sycomore^{++} in presence of a peak of load.

Fig. 2: Scalability of Bitcoin, Sycomore and Sycomore^{++} (overload threshold $\Gamma = 90\%$, underload threshold $\gamma = 0\%$) and Reactivity of both Sycomore and Sycomore^{++} ($\Gamma = 90\%$, $\gamma = 10\%$).

to the number of leaf blocks, and thus to f_{req}, Sycomore^{++} exhibits an optimal behavior regarding confirmed transactions, i.e., $\forall f_{\text{req}}$, the transaction confirmation rate equals the transaction creation rate. For the sake of comparison, confirming 160 txs/tick with the simulator (and as previously said we cannot stress more the simulator) means confirming $6,400$ txs/mn in the real life.

Figure 2b shows the average number of pending transactions at miners, that is the average number of transactions that accumulate at miners before being embedded in blocks. It clearly shows that for both Bitcoin and Sycomore, this number linearly increases with the transaction submission rate once it exceeds 10txs/tick since this corresponds to the global inter-block creation delay. In contrast, by adapting the number of created blocks to f_{req}, Sycomore^{++} drastically reduces the average number of pending transactions. For example, for $f_{\text{req}} = 10$ txs/tick, this number is equal to 432 transactions, and for $f_{\text{req}} = 160$ txs/tick, it is equal to $1,957$ transactions, compared to $154,000$ ones in both Bitcoin and Sycomore.

Figure 2c illustrates the lost energy as a function of f_{req}. In Bitcoin, since the inter-block delay, the number of miners, and the difficulty do not vary during each experiment, the same amount of energy is lost regardless of f_{req}. While this setting also applies to Sycomore, the fact that the SYC-DAG becomes larger with increasing values of f_{req} gives rise to a uniform distribution of the global computational power over the leaf chains, and thus decreases miners' competition. Thus less work is wasted w.r.t Bitcoin. Regarding Sycomore^{++}, for increasing values of f_{req}, the SYC-DAG becomes larger and blocks are created faster (as the mining difficulty adapts to the SYC-DAG structure), which allows Sycomore^{++} to reach an optimal number of leaf chains more quickly than Sycomore does. As a consequence, we get a better parallelism of miners' work, and thus a drastic reduction of energy loss.

Figure 2d illustrates the average transaction latency as a function of f_{req}. Recall that the transaction latency measures the time elapsed between the instant at which a transaction is submitted to the network by the user and the time it becomes confirmed in the ledger. In contrast to all the other experiments, transaction latency has been measured as follows: transactions are submitted at f_{req} for a while, then f_{req} is set to 0, and simulations stop once all the submitted transactions have been confirmed. The first observation is that, in Bitcoin, once $f_{\text{req}} \geq 10$ txs/tick, transaction latency linearly increases with f_{req}, which clearly corroborates both Figures 2a and 2b. Regarding Sycomore, the loss of performance w.r.t Bitcoin is due to the fact that blocks in all the sibling chains are not necessarily fully loaded, which delays accordingly transaction latency. On the other hand, Sycomore^{++} enjoys an average constant latency, which is equal to 50 ticks regardless of f_{req}. Essentially, the more transactions are submitted, the more blocks are filled until the optimal shape of the graph is reached. Results shown in Figures 2a, 2b combined with these results clearly demonstrate the exemplary behavior of Sycomore^{++}: transactions are confirmed at the rate at which they are submitted by clients to the system, and their latency is constant whatever the submission creation rate, meaning that users can safely predict the time at which their transaction, if

valid, will be deeply confirmed in Sycomore^{++}.

D. Reactivity Study

This section aims at assessing the capacity of both Sycomore and Sycomore^{++} SYC-DAG to react to sudden and abrupt fluctuations in the creation transaction rate. [3]

1) Experiments setting: As briefly presented in Section III, when f_{req} shrinks, the SYC-DAG reacts by progressively decreasing the under loaded sibling chains, and thus the number of created blocks. Thus each merge divides by almost two the number of blocks that will be subsequently created. By the randomness of transaction identifiers, if one chain becomes under loaded, then soon after all, the chains will become under loaded too, and thus merges will occur in cascade. Initially, $f_{\text{req}} = 100$ txs/tick during 10 ticks to mimic a transaction peak load, and then at tick $t = 12$, $f_{req} = 0$ txs/tick.

2) Experiments results: Figure 2e illustrates the reactivity of both Sycomore and Sycomore^{++} in presence of a load peak (illustrated by the red constant function from $t = 1$ to $t = 11$ ticks at $f_{\text{req}} = 100$txs/tick). Both Sycomore and Sycomore^{++} initially undergo a series of splits, and then progressively move on to a series of merge up to converging to a single chain of blocks. Sycomore^{++} differs from Sycomore in its rapidity to split and merge: Sycomore^{++} succeeds in coping with the load pick 75% faster than Sycomore does, and 33% faster than Sycomore to cope with the sudden shrink of load. It is worthwhile to observe that those results combined with the one observed in Section V-C, assess the capability of both Sycomore and Sycomore^{++} to meet Properties P1 and P4.

E. Adversarial environment

This section measures the impact of high transmission delays on the number of forks and the time it takes for Bitcoin, Sycomore and Sycomore^{++} to resolve them. This section supposes that all miners are honest and thus do not design adversarial strategies to create forks (adversarial behaviors are studied in Section V-F). We suppose that simultaneous events are not possible. Thus if transmission delays are null, fork can never occur (once a miner receives a block b that would be appended to the same leaf block as its own currently created block b', it does not broadcast b'). When transmission delays increase, miners will broadcast their blocks before detecting the presence of concurrent ones, giving rise to forks. Let Δ be the constant transmission delay on the network. Let t_b and $t_{b'}$ be the instant at which the two concurrent blocks b and b' are respectively broadcast. Forks can occur only if Δ is greater than the time elapsed between t_b and $t_{b'}$.

1) Experiment settings: We vary the overload threshold Γ from 90% to 100% and set the underload threshold γ to 0%, so that merge do not happen (for $\Gamma = 100\%$, splits never happen and thus the SYC-DAG reduces to Bitcoin's chain). f_{req} is set to 160txs/tick to provoke splits. Δ ranges from 0 (no fork) to 5 ticks. It is important to observe that $\Delta = 5$ ticks is very large compared to the average time needed to create a block (i.e., 10 ticks). The reason is that we want to stress the system under constant and very high submission rates to provoke splits, and large transmission delays to study their impact on the occurrence and resolution of forks.

[3]We omit Bitcoin from this evaluation since Bitcoin chain does not adapt to transaction demand.

	$\Delta = 0$		$\Delta = 0.1$		$\Delta = 1$		$\Delta = 5$	
	f	t_r	f	t_r	f	t_r	f	t_r
Bitcoin	0	0	0.5	1.4	0.9	10.8	2.3	36.6
Sycomore	0	0	0	0	0	0	0.6	11.7
Sycomore^{++}	0	0	0	0	0	0	1.1	6.1

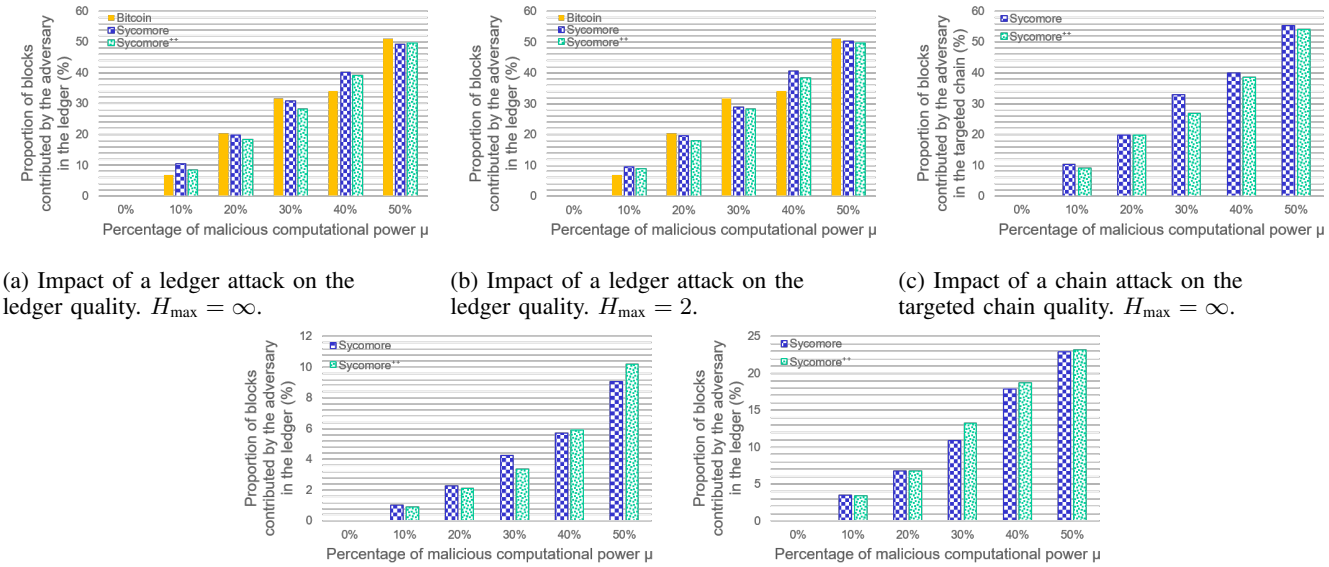
TABLE I: Average number f of forks and average time to resolve one fork (t_r) as a function of the network delay (ticks).

2) Experiment results: The main results drawn from our experiments appear in Table I, which shows the impact of Δ on the the number of forks f and their resolution time t_r (in ticks). Forks are alternative stories, that is having n forks in a simulation means having $n + 1$ alternative ledgers. The fork resolution time t_r is equal to the time elapsed between the creation of an alternative chain (Bitcoin) or SYC-DAG (Sycomore and Sycomore^{++}) and the instant at which a ledger has the best confirmation level of the genesis block w.r.t the others ledgers (see Rule 1, Section III). As can be observed, the number of forks f increases with Δ. As Sycomore and Sycomore^{++} differ in their capacity to continuously adjust the difficulty to the actual number of leaf blocks, their tolerance to fork occurrence is different: decreasing (resp. increasing) the mining difficulty reduces (resp. enlarges) the standard deviation between any two blocks b and b' creation times, and therefore impacts the probability with which $\Delta > |t_b - t_{b'}|$ holds or not. On the other hand, as blocks are created faster, fork resolution takes less time in Sycomore^{++} than in Sycomore. Hence, if sellers adopt the same rule as in Bitcoin to wait for a given period of time T before sending their goods to buyers, we clearly see that both Sycomore and Sycomore++ drastically reduce T (i.e., $T/3$ for Sycomore and $T/6$ for Sycomore^{++} w.r.t Bitcoin for $\Delta = 5$ ticks).

F. Adversarial strategies

a) Experiments setting: Both attacks share the same experiment settings. We set $f_{\text{req}} = 40$ txs/tick to provoke splits. The proportion of computational power μ owned by the adversary ranges in the interval $[0\%-50\%]$. Operationally, we divide miners into two groups, the honest and malicious groups, and allocate each group with a proportion of the total computational power W, i.e., $W(1-\mu)$ for the honest group and $W\mu$ for the malicious one. Experiments are run with $H_{max} = \infty$ and $H_{max} = 2$.

This section studies the resilience of Bitcoin, Sycomore and Sycomore^{++} in presence of adversarial strategies that could hinder the chain quality property as defined in [10]. The chain quality property states that, in Bitcoin, the adversary may control at most a $\mu/(1-\mu)$ percentage of the blocks in the chain, where μ represents the ratio of the network hashing power owned by the adversary. As briefly explained in Sections III and IV, both Sycomore and Sycomore^{++} aim



(a) Impact of a ledger attack on the ledger quality. $H_{\max} = \infty$.

(b) Impact of a ledger attack on the ledger quality. $H_{\max} = 2$.

(c) Impact of a chain attack on the targeted chain quality. $H_{\max} = \infty$.

(d) Impact of a chain attack on the ledger quality. $H_{\max} = \infty$.

(e) Impact of a chain attack on the ledger quality. $H_{\max} = 2$

Fig. 3: Impact of the ledger and chain attacks

at guaranteeing that miners can neither foresee nor choose the leaf chain to which their block will be appended prior to having irremediably completed the construction of their block's header (Property P3). The reason is to prevent an adversary from devoting all its computational power to the growing of a specific chain. This means that the only way for the adversary to target a specific chain is to repeatedly generate blocks until a valid header for the targeted chain is produced. This process is computationally intensive and thus, the objective of these experiments is to determine how much mining power the adversary should exert to have an effective impact on targeted chains. This has led us to design and implement the following two attacks. In the first one, called *ledger attack*, the attacker tries to undermine the whole ledger quality, i.e., tries to maximize the proportion of blocks it contributed in Bitcoin chain, and in each chain of the SYC-DAG in both Sycomore and Sycomore^{++}. In the second one, called *chain attack*, the adversary targets a specific chain of the SYC-DAG and tries to maximize the proportion of blocks it contributed within this chain. [4] This requires for the adversary to only keep the blocks they have mined that match the targeted leaf chain. Note that the honest miners feed all leaf chains equally, including the one targeted by the adversary. The impact of the readjustment period $H_{\max}$ is noticeable in the chain attack. Indeed, as discussed in Section III, at each readjustment period, i.e., each time the length of the ledger has been increased by $H_{\max}$ w.r.t to the previous re-adjustment, (*i*) the mining difficulty is recomputed to adapt to the actual computational power of the system and (*ii*) late leaf chains catch up to $H_{\max}$ if necessary. In the latter case this means that if the chain targeted by the adversary is among the first ones to reach $H_{\max}$, then the adversary (and the honest miners) will not be able to contribute blocks on the targeted chain (any block appended to these fast leaf chains will be ignored by the honest miners as long as the other chains have not caught up). So to increase the speed at which all the leaf blocks caught up, the adversary contributes blocks on these late leaf blocks, so that it will be able to contribute on the targeted chain quicker. It is important to note that all the blocks contributed by the adversary are all valid otherwise they would not appear in the ledger. However, they possibly favour particular transactions submitted by the adversary, or in contrast do not contain transactions the adversary wishes to exclude from the ledger.

1) Ledger attack: Figures 3a and 3b illustrate the impact of the ledger attack in Bitcoin, Sycomore and Sycomore^{++} distributed ledgers. The main observation drawn from both figures is the fact that the ledger quality property [10] holds in Bitcoin, Sycomore and Sycomore^{++}: the adversary cannot control more than $\mu/(1-\mu)$ percent of the blocks in the ledgers. Furthermore the impact of $H_{\max}$ value is negligible in both Sycomore and Sycomore^{++} since the adversary has no better strategy than appending the maximum number of blocks on each chain of the SYC-DAG.

2) Chain attack: Figures 3c, 3d and 3e illustrate the impact of the chain attack on the quality of the targeted chain (Figure 3c) and on the quality of the ledger (Figures 3d and 3e). We have implemented the chain attack as follows: the malicious group of miners focuses on the leaf chain with the lowest label (this could be any existing leaf label) and only appends blocks to it, which requires for the adversarial group to discard all the blocks they have mined that do not

[4]Note that the chain attack does not make sense in Bitcoin.

match the lowest label. The main observation drawn from Figure 3c is the fact that in both Sycomore and Sycomore^{++}, the chain quality holds: the adversary cannot control more than $\mu/(1-\mu)$ percent of the blocks in the targeted chain. This figure also illustrates that in both Sycomore and Sycomore^{++}, the computational power is equally distributed on the SYC-DAG leaf chains.

Figures 3d and 3e show the very low impact of the chain attack on both Sycomore and Sycomore^{++} quality. For instance, if the adversary has 50% of the hashing power, then it will control no more than 10% of the blocks in the honest players's ledger. It is interesting to see the impact of $H_{\max}$ value on the ledger quality: when $H_{\max} = \infty$, the adversary continuously tries to append its contributed blocks to its targeted chain at the expense of throwing away all its blocks that do not fit this chain. On the other hand, when $H_{\max} = 2$, the adversary must periodically feed the other chains when its targeted chain has been increased by 2 before all the other ones (actually this is often the case since both the adversary and the honest miners contribute to this targeted chain). As a consequence, the percentage of blocks contributed by the adversary in the ledger augments in both Sycomore and Sycomore^{++}, while completely satisfying the ledger quality property [10].

VI. Conclusions

In this paper we have presented an advanced experimental study to assess the properties of three permissionless PoW-based distributed ledgers, Bitcoin, Sycomore, and Sycomore^{++}. Both Sycomore and Sycomore^{++} organize blocks along a particular directed acyclic graph, whose structure evolve according to the actual load of the system. Sycomore^{++} drastically improves upon Sycomore by continuously adapting the mining difficulty to the graph structure. Experimental results show impressive results (compared to both Bitcoin and Sycomore) in terms of scalability, energy loss, reactivity, resilience, and quality of the distributed ledger. As future work, we intend to analyse the computational cost of adversarial strategies in these distributed ledgers in presence of transient network partitions.

References

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

[2] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," 2016.

[3] B. M. David, P. Gazi, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *EUROCRYPT*, 2018.

[4] J. Chen and S. Micali, "Algorand: A secure and efficient distributed ledger," *Theor. Comput. Sci.*, vol. 777, pp. 155–183, 2019.

[5] L. Aştefănoaei, P. Chambart, A. Del Pozzo, T. Rieutord, S. Tucci-Piergiovanni, and E. Zălinescu, "Tenderbake - A Solution to Dynamic Repeated Consensus for Blockchains," in *4th International Symposium on Foundations and Applications of Blockchain 2021 (FAB 2021)*.

[6] J. Poon and T. Dryja, *The bitcoin lightning network*, 2016. [Online]. Available: https://lightning.network/lightning-network-paper.pdf

[7] C. Burchert, C. Decker, and R. Wattenhofer, "Scalable funding of bitcoin micropayment channel networks," in *SSS*, 2017.

[8] A. Ranchal-Pedrosa, M. G. Potop-Butucaru, and S. T. Piergiovanni, "Scalable lightning factories for bitcoin," *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, 2019.

[9] E. Anceaume, A. Guellier, R. Ludinard, and B. Sericola, "Sycomore: A permissionless distributed ledger that self-adapts to transactions demand," in *Proceedings of the IEEE 17th International Symposium on Network Computing and Applications (NCA)*, 2018.

[10] 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.

[11] C. Kaligotla and C. M. Macal, "A generalized agent based framework for modeling a blockchain system," in *Proceedings of the Winter Simulation Conference (WSC)*, 2018.

[12] P.-Y. Piriou and J.-F. Dumas, "Simulation of stochastic blockchain models," in *Workshop on Blockchain Dependability organized with the 14th European Dependable Computing Conference*, 2018.

[13] M. Alharby and A. van Moorsel, "Blocksim: A simulation framework for blockchain systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 46, pp. 135–138, 01 2019.

[14] E. Rosa, G. D'Angelo, and S. Ferretti, "Agent-based simulation of blockchains," *ArXiv*, vol. abs/1908.11811, 2019.

[15] C. Faria and M. Correia, "Blocksim: Blockchain simulator," in *IEEE International Conference on Blockchain (Blockchain)*, 2019.

[16] M. Bottone, F. Raimondi, and G. Primiero, "Multi-agent based simulations of block-free distributed ledgers," 2018.

[17] S. Popov, "The tangle. iota white paper," 2015. [Online]. Available: https://iota.org/

[18] U. Wilensky, "Netlogo." Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL., 1999. [Online]. Available: http://ccl.northwestern.edu/netlogo/

[19] L. Baird, "The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance," Tech. Rep., 2016. [Online]. Available: http://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf

[20] A. Churyumov, "ByteBall : A decentralized system for storage and transfer of value," 2017. [Online]. Available: https://byteball.org/Byteball.pdf

[21] G. Bu, Ö. Gürcan, and M. Potop-Butucaru, "G-IOTA: fair and confidence aware tangle," *CoRR*, vol. abs/1902.09472, 2019. [Online]. Available: http://arxiv.org/abs/1902.09472

[22] Y. Sompolinsky and A. Zohar, "Accelerating bitcoin's transaction processing. fast money grows on trees, not chains," *IACR Cryptology ePrint Archive*, vol. 2013, 2013.

[23] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "Spectre: A fast and scalable cryptocurrency protocol," *IACR Cryptol. ePrint Arch.*, p. 1159, 2016.

[24] Sycomore^{++}, "Source code," https://anonymous.4open.science/r/Sycomorepp-412D.

[25] N. Lagaillardie, M. A. Djari, and O. Gurcan, "A computational study on fairness of the tendermint blockchain protocol," *Information*, vol. 10, no. 12, 2019. [Online]. Available: https://www.mdpi.com/2078-2489/10/12/378

[26] O. Gutknecht and J. Ferber, "The madkit agent platform architecture," in *Proceedings of the International Workshop on Infrastructure for Multi-Agent Systems: Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, 2000.

[27] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec, "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science*, ser. Communications in Computer and Information Science, I. I. Ivanov, M. van Sinderen, F. Leymann, and T. Shan, Eds. Springer International Publishing, 2013, vol. 367, pp. 3–20.

[28] B. H. Distribution, 2020. [Online]. Available: https://blockchair.com/bitcoin/charts/hashrate-distribution

[29] J. Göbel and A. Krzesinski, "Increased block size and bitcoin blockchain dynamics," in *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*, 2017, pp. 1–6.