Simulation study of Sycomore ++ , a self-adapting graph-based permissionless distributed ledger

Aimen Djari, Emmanuelle Anceaume, Sara Tucci-Piergiovanni

Simulation study of Sycomore^{++}, a 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.**

***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 work 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] has been the first Nakamoto-style Proof-of-Work protocol that addresses Bitcoin's scalability issues by making its graph-structure dynamically adapt to fluctuations in transaction submission rates. Specifically, when the last blocks appended to a given chain $\mathcal{C}$ of the graph exceed some maximal load 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 load 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. Motivation for this Work

By dynamically adapting the number of chains of the graph to the actual transaction load of the network, one might expect that Sycomore would guarantee an almost optimal scalability in terms of confirmed transactions per second (TPS). Actually, the rate at which transactions are confirmed fully depends on the mining difficulty, i.e., the difficulty to create the next block. To cope with variations of the network computational power (i.e., hashrate of the system), the mining difficulty is periodically readjusted to guarantee 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 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 hashrate 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, which in the worst case will require a power consumption equal to Bitcoin's one. Transaction throughput will drastically decreases until the next readjustment of the difficult takes place.

B. Contributions of this Work

This work presents a twofold contribution. First we present Sycomore^{++}, a scalable Nakamoto-style Proof-of-Work protocol that solves the critical issues mentioned above. Sycomore^{++} inherits the main mechanisms of Sycomore, while adding a new adaptive adjustment mechanism that continuously adapts the number of confirmed transactions per second (TPS) to the actual rate at which transactions are submitted to the network. Secondly, we propose fine-grained simulations to evaluate the capacity of Nakamoto-style Proof-of-Work protocols, i.e., Bitcoin, Sycomore and Sycomore^{++}, to scale. We have implemented these protocols on an agent-based simulator and we have compared the scalability of these protocols in the presence of large communication delays and sudden and substantial variations of the system workload demand. These experiments show that Sycomore^{++} guarantees an optimal throughput, i.e., transactions are confirmed at the rate at which they are submitted by clients, whatever their submission rate, and transaction latency is constant. One of the consequences is that users can safely predict the time at which their (valid) transaction will be confirmed in Sycomore^{++}. Furthermore Sycomore^{++} shows a drastic reduction of "lost" hashrate w.r.t Bitcoin since concurrent blocks are not necessarily in a fork situation. Finally, we have designed sophisticated attacks that an adversary may elaborate to hinder the ledger quality property. Experiments show that Sycomore^{++} is resistant to these adversarial behaviors.

Finally, the main shortcoming of this work is to have specifically focused on Bitcoin, Sycomore and Sycomore^{++}, and not on other DAG-based PoW protocols. We have started the implementation of both Ghost [11] and Spectre [12] in our simulator to analyze the impact of their transaction handling and to compare their scalability and resilience properties with Bitcoin, Sycomore and Sycomore^{++}'s ones.

The remaining of the paper is organized as follows. Section II discusses related work on simulation and graph-based distributed ledgers. Sycomore^{++} is presented in Section III together with an analytical performance study. Section IV highlights the main lessons learnt from the experiments we have conducted. Finally, Section V 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. [13] 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 hashrate, and a large range of transaction submission rates, network delays, and adversarial strategies are considered. Piriou et al. [14] 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. [15], Rosa et al. [16] and Faria et al. [17] 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. [18] present a simulation model for Tangle-like DAG ledgers [19] aiming at studying the grow of the graph of transactions. The simulation model is instrumented on the NetLogo [20] 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 [21], ByteBall [22], and Iota [19], [23] 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 [11] and Spectre [12] 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 [12] 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 [12]. In Sycomore [9] 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. Sycomore^{++}

Sycomore^{++} is a Nakamoto-style Proof-of-Work protocol. It builds a permisionless cryptocurrency ledger whose structure is a dedicated balanced directed acyclic graph of blocks, called SYC-DAG, introduced in Sycomore [9]. Sycomore^{++} enjoys a set of properties inherited from Sycomore, and thanks to a new adaptive adjustment mechanism, it guarantees an optimal scalability: the rate at which transactions are deeply

confirmed equals their submission rate in the system. To make the paper self-contained, we highlight the design principles of Sycomore, and then present the new adaptive adjustment mechanism of Sycomore^{++}.

A. Overview of Sycomore

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

Property P1: Dynamic adaptation of the structure of the ledger. This has been implemented by introducing the notion of *splittable* and *mergeable* blocks, which are a dynamic response to respectively a rise or a drop in the number of transactions inserted in the last blocks appended to the SYC-DAG. Both notions refer to block load, i.e., the ratio between its number of bytes and its maximal load (e.g., 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 ($c_{\min}$ 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, called sibling chains, such that the first block of each of the two chains refers to b. Conversely, when the block load decreases Sycomore progressively reduces the number of chains in the SYC-DAG. Hence, 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 merged chain. Any block that is neither mergeable nor splittable is *regular*. As argued in [9], everyone, and in particular miners, can 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). Note that for the interested reader, an example of a SYC-DAG is provided in the appendix.

Property P2: Balanced partitioning of transactions. It aims at fully exploiting the gain brought by sibling chains, and is implemented by introducing the notion of *label*. A label is a binary string, and characterizes the common prefix of the identifiers (i.e., fingerprints) 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 (affectation of a block to a given chain is described in the next paragraph). 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 are labelled with the empty string ε (this reflects Bitcoin's behavior). Now all the blocks of two sibling chains, appended to a splittable block b inherit b's label extended with 0 and 1 respectively, and will only contain transactions whose identifier is prefixed by b's label extended with 0 (resp. extended with 1). Conversely, all the blocks that belong to a merged chain inherit the largest common prefix of its predecessor labels. As transactions' identifiers can be considered as random bit strings, transactions are evenly partitioned over sibling chains, and cannot appear in more than one block.

Property P3: Unpredictability of the predecessor. This property 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 mined by any miner u contains, among different pieces of information, a set of $(c + s)$ commitment tuples $\{\ldots, (H(b^{\ell_j}), \ell'_j, m^{\ell'_j}), \ldots\}$ that *(i)* acknowledges or commits u's local view $\mathcal{L}_u$ of the SYC-DAG, and *(ii)* characterizes b's predecessor. If $\mathcal{L}_u$ contains c leaf blocks $b^{\ell_1}, \ldots, b^{\ell_c}$ at the time u starts b's creation process, and among them, s of them are splittable, then for $j \in [\![1, c]\!]$, $H(b^{\ell_j})$ 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 1). 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 [9]. 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: chain fairness. This property aims at guaranteeing that with high probability, all the leaf chains of the SYC-DAG grow at the same speed. It 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: low probability of forks 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 current number of leaf blocks in the SYC-DAG.

Based on the above description, we have:

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 $\forall i \in [\![1, n]\!]$, $\mathcal{C}^{\ell_i}$ is labelled ℓ_i (note that several chains in $\mathcal{P}$ may be assigned the same label) and the following three properties hold:

$$\forall \mathcal{C}^{\ell_i} \in \mathcal{P}, \forall k \in [\![0, |\ell_i| - 1]\!], \mathcal{C}^{\ell_i^{\downarrow k}} \in \mathcal{P} \quad (1)$$

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

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

Similarly to all Nakamoto-style PoW protocols, 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 1 (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.*

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.

B. Adaptive Adjustment of the Difficulty

As motivated in the introduction, Sycomore^{++} improves upon Sycomore by continuously adapting the block creation difficulty D to the actual number of leaf chains of the SYC-DAG. This guarantees that whatever the structure of the SYC-DAG, a constant inter-block creation delay is maintained on any of its chains. This adaptive adjustment does not replace the periodic readjustment to the total hashrate of the system present in any Nakamoto-style PoW protocols. The former adapts the difficulty D to the structure of the SYC-DAG while the latter adapts the difficulty D to the total hashing power of the system. Specifically, periodically the mining difficulty D is 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 of leaf blocks. This periodic adjustment takes place every time t_{adj} 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 $h = 0 \bmod H_{\max}$, with $H_{\max} = 2016$. 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. If D represents the current difficulty, then at time t_{adj}, the new difficulty D_{adj} is calculated as $D_{adj} = (d_n/d)D$, where d_n represents 14 days, and d is the time needed to create and append 2016 blocks on each chain of the SYC-DAG. 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. In addition to this periodic readjustment, the difficulty D is continuously adjusted to fit the current structure of the SYC-DAG in order to cope with fluctuations in the creation transaction rate (either due to normal peaks or drops of activities or due to some adaptive adversarial strategies). Hence, whenever the current number of leaf chains changes, the new difficulty D is calculated as $D = D_{adj}/c$, where c is new number of leaf chains, and D_{adj} computed as above.

Lemma 1 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. This behavior is confirmed by the experimental evaluation presented in Section IV.

Lemma 1. *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. The interested reader is invited to read the proof in the appendix. □

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

Lemma 2. *Given a ledger $\mathcal{L}^\star_v$ 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 $\mathcal{C}_i$, $i \in [\![1, c]\!]$, 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. The interested reader is invited to read the proof in the appendix. □

Lemma 3. *For any correct node u, $\mathcal{L}^\star_u$ does not contain double-spending transactions.*

Proof. The interested reader is invited to read the proof in the appendix. □

IV. Simulation Study

This section presents the agent-based simulation study we have conducted to assess performance of Nakamoto-style Proof-of-Work protocols and resiliency to sophisticated attacks. The source codes of these protocols 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 develop distributed ledger protocols. It 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 work. All the experiments have been run on Grid'5000, a large-scale and flexible test-bed for experiment-driven research [27].

B. Simulation Model

1) Block creation model: For straightforward reasons, miners do not solve proof-of-works. Hence, 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 hashrate W_u. W_u is a fraction of the hashrate 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 is the security parameter of the Proof-of-Work and D is the difficulty. 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. 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 fingerprint (since the PoW nonce ν is not computed) and each leaf block of the SYC-DAG. 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].

2) Common parameters of the simulations:

- The **block capacity**, i.e., the maximal number of transactions in a block, 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, reducing the block capacity does not affect protocols behavior.
- A transaction is **confirmed** when the block b this transaction belongs to has a confirmation level equal to $k = 6$. Note that differently from Bitcoin, in both Sycomore and Sycomore^{++}, these blocks can belong to different chains of the SYC-DAG, as long as these blocks form a path of commitment down to block b (see Section III-A).
- $\mathbf{c}_{\text{min}}$ is set to 1. Impact of c_{min} on the structure of the SYC-DAG and its performances is left for future work.
- For each experiment, we have run sufficiently many simulations to get a confidence interval equal to $5 \pm \%$.

C. Scalability Study

We evaluate the transaction confirmation rate, the transaction latency (i.e., 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) under high transaction submission rates. The energy lost by each protocol is also measured. It is the sum for each miner u and for each created block b not appended to the ledger of the time spent working on b times the hashrate cp_u. In this section, we assume that forks do not occur.

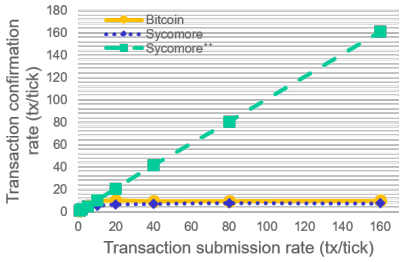
1) Experiment setting: The overload threshold Γ, which conditions the SYC-DAG splitting, varies from 90% to 100%. Note that when $\Gamma = 100\%$, splits never occur and thus both Sycomore and Sycomore^{++} reduce to Bitcoin. The submission rate of transactions f_t varies from 1 to 160txs/tick. We tune the PoW parameters to get one block mined every 10 ticks in expectation. Thus in Bitcoin $f_t = 10$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 $f_t > 10$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 $f_t = 160$txs/tick. Anyway, $f_t = 160$txs/tick already severely stresses the three protocols.

2) Experiment results: Note that in all the plots in Figure 1, points are linked together with lines for readability reasons.

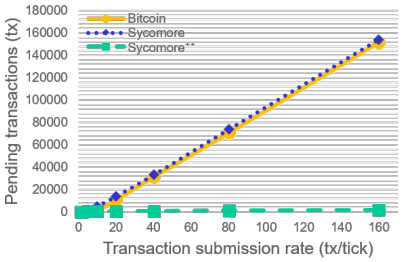
Figure 1a shows the confirmation rate of transactions as a function of their submission rate f_t. The main observation regarding Bitcoin and Sycomore is that whatever the network hashrate, no more than 10txs/tick are confirmed, which illustrates the impact of the globally constant inter-block delay (i.e., a block is mined every 10 ticks in average). Sycomore shows slightly worse results than Bitcoin 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 to the number of leaf blocks, and thus to f_t, Sycomore^{++} exhibits an optimal throughput: $\forall f_t$, the transaction confirmation rate equals f_t.

Figure 1b shows 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 f_t once f_t exceeds 10txs/tick as this corresponds to the global inter-block creation delay. In contrast, by adapting the number of created blocks to f_t, Sycomore^{++} drastically reduces the average number of pending transactions. For example, for $f_t = 10$txs/tick, this number is equal to 432 transactions, and for $f_t = 160$txs/tick, it is equal to $1,957$ transactions, compared to the $154,000$ ones in both Bitcoin and Sycomore.

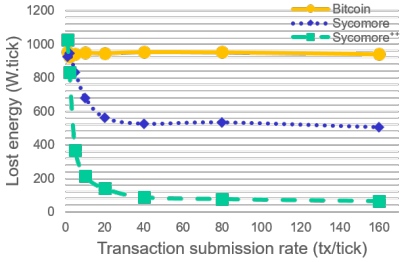
Figure 1c illustrates the lost energy as a function of f_t. In Bitcoin, as the inter-block delay, the number of miners, and the difficulty do not vary during each experiment, the same amount of energy is lost regardless of f_t. While this setting also applies to Sycomore, the fact that the SYC-DAG becomes larger with increasing values of f_t gives rise to a uniform distribution of the total hashrate 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_t, 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 faster than Sycomore does. We get a better parallelism



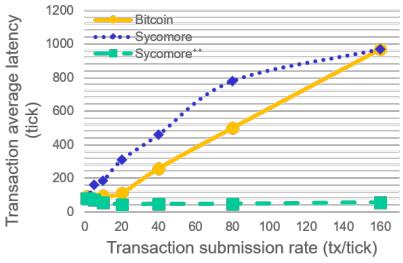
(a) Transaction confirmation rate as a function of the transaction submission rate.



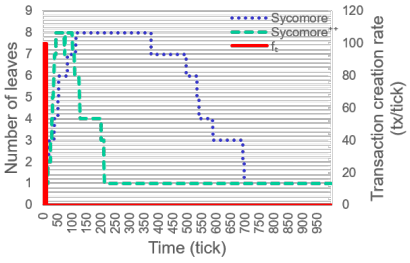
(b) Number of pending transactions as a function of their submission rate.



(c) Lost energy as a function of their submission rate.



(d) Transaction average latency as a function of their submission rate.



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

	$\Delta = 0$		$\Delta = 0.1$		$\Delta = 1$		$\Delta = 5$	
	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
Syc.	0	0	0	0	0	0	0.6	11.7
Syc.$^{++}$	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).

Fig. 1: Scalability of the three protocols ($\Gamma = 90\%$, $\gamma = 0\%$) and Reactivity of both Sycomore and Sycomore^{++} ($\Gamma = 90\%$, $\gamma = 10\%$).

of miners' work, and thus a drastic reduction of energy loss.

Figure 1d illustrates the average transaction latency as a function of f_t. The transaction latency measures the time elapsed between the instant at which a transaction is submitted to the network 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_t for a while, then f_t is set to 0, and simulations stop once all the submitted transactions have been confirmed. In Bitcoin, once $f_t \geq 10$ txs/tick, transaction latency linearly increases with f_t, which clearly corroborates both Figures 1a and 1b. 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_t. Essentially, the more transactions are submitted, the more blocks are filled until the optimal shape of the graph is reached.

Results shown in Figures 1a, 1b and 1d clearly demonstrate the exemplary behavior of Sycomore^{++}: transactions are confirmed at the rate at which they are submitted by clients, and their latency is constant whatever their submission rate, meaning that users can safely predict the time at which their transaction, if valid, will be confirmed in Sycomore^{++}.

D. Reactivity Study

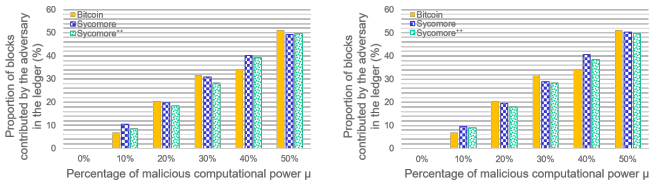
This study aims at assessing the capacity of both Sycomore and Sycomore^{++} to react to sudden and abrupt fluctuations of f_t. We omit Bitcoin from this evaluation since Bitcoin chain does not adapt to transaction demand.

1) Experiments setting: As presented in Section III, when f_t shrinks, the SYC-DAG reacts by progressively decreasing the under loaded sibling chains. 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, and thus merges will occur in cascade.

2) Experiments results: Figure 1e illustrates the reactivity of both protocols in presence of a transaction load peak (illustrated by the red constant function from $t = 1$ to $t = 11$ ticks with $f_t = 100$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 react: 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. Observe that those results combined with the one in Section IV-C, assess the capability of these protocols to meet Properties P1 and P4.

E. Adversarial environment

We study the impact of high communication delays on the fork number and the time it takes for the three protocols to resolve them. We suppose that all miners are honest and thus do not design adversarial strategies to create forks (this is studied in Section IV-F). Simultaneous events are not possible, thus if communication 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 communication delays increase,



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

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

Fig. 2: Impact of ledger attacks

miners broadcast their blocks before detecting the potential presence of concurrent ones, giving rise to forks. Let Δ be the communication delay, and t_b and $t_{b'}$ be the instants at which two concurrent blocks b and b' are respectively broadcast, then forks can occur only if $\Delta > |t_b - t_{b'}|$.

1) Experiment settings: The overload threshold Γ varies from 90% to 100% and the underload threshold γ equals 0%, so that merges do not happen. $f_t = 160$txs/tick to provoke numerous splits. Δ ranges from 0 (no fork) to 5 ticks. Note that $\Delta = 5$ ticks is very large compared to the average block creation time (i.e., 10 ticks). Indeed, 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.

2) Experiment results: Table 1 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 (see Rule 1). Clearly, the number of forks f increases with Δ. As Sycomore and Sycomore^{++} differ in their capacity to adjust the difficulty to the actual number of leaf blocks, fork occurrence differs: decreasing (resp. increasing) the mining difficulty reduces (resp. enlarges) the standard deviation between any two blocks b and b' submission 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 is quicker 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, both Sycomore and Sycomore++ drastically reduce T w.r.t. Bitcoin: $T/3$ for Sycomore and $T/6$ for Sycomore^{++} for $\Delta = 5$ ticks.

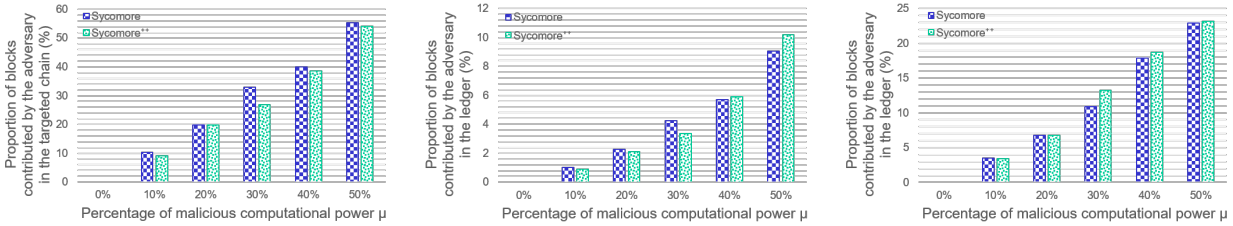
F. Adversarial strategies

We now study the resilience of the three protocols in presence of adversarial strategies that aims at hindering the chain quality property [10]. This property, defined in the context of Bitcoin, states that 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. In Sycomore and Sycomore^{++}, 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 (see Property P3). This prevents an adversary from devoting all its hashrate to the growing of a specific chain. Thus 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. We have designed and implemented two attacks. In the *ledger attack*, the attacker tries to undermine the whole ledger quality, i.e., maximize the proportion of blocks it has contributed in Bitcoin chain, or in each chain of the SYC-DAG. In the *chain attack*, the adversary targets a specific chain of the SYC-DAG and tries to maximize the proportion of blocks it has contributed in this chain. This requires for the adversary to only keep blocks that match the targeted leaf chain. Honest miners feed all leaf chains equally, including the one targeted by the adversary. Note that the chain attack does not make sense in Bitcoin. It is important to note that blocks contributed by the adversary are 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.

a) Experiments setting: Both attacks share the same experiment settings. We set $f_t = 40$ txs/tick to provoke splits. The proportion of hashrate μ owned by the adversary ranges in $[0\%, 50\%]$. Operationally, we divide miners into two groups, the honest and malicious ones, 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.

1) Ledger attack: Figures 2a and 2b illustrate the impact of the ledger attack for the three protocols. The main observation drawn from the plots is the fact that the ledger quality property [10] always holds in Bitcoin, Sycomore and Sycomore^{++} whatever the proportion μ of malicious hashrate: 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: The impact of the chain attack on the quality of the targeted chain is illustrated in Figure 3a. This attack is implemented as follows: the malicious group 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 match the lowest label. Both Sycomore and Sycomore^{++} guarantee the chain quality: the adversary cannot control more than $\mu/(1-\mu)$ percent of the blocks in the targeted chain. The impact of the chain attack on the ledger quality is shown in Figures 3b and 3c. Both figures assess the very low impact of this attack on both Sycomore and Sycomore^{++}'s quality. For instance, if the adversary owns



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

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

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

Fig. 3: Impact of chain attacks

50% of the hashing power, then it will control no more than 10% of the blocks in the honest players's ledger (see Figure 3b). The impact of $H_{\max}$ value on the ledger quality is noticeable: 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. Now, 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. As a consequence, the percentage of blocks contributed by the adversary in the ledger augments in both Sycomore and Sycomore^{++}. Note however that the ledger quality property [10] still holds.

V. Conclusions

We have presented an advanced experimental study to assess the properties of three permissionless PoW-based distributed ledgers under high or chaotic submission transaction rate, and adversarial environments. Experimental results show impressive results of Sycomore^{++} compared to both Bitcoin and Sycomore in terms of scalability, energy loss, reactivity, resilience, and quality of the distributed ledger. We are currently developping Ghost [11] and Spectre [12] in our simulator so that to provide a comprehensive experimental study of DAG-based protocols, and will focus on adversarial strategies in presence of transient network partitions.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system." [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," 2016. [Online]. Available: https://atrium.lib.uoguelph.ca/xmlui/handle/10214/9769

[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 *Symposium on Foundations and Applications of Blockchain (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 *Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 2017.

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

[9] E. Anceaume, A. Guellier, R. Ludinard, and B. Sericola, "Sycomore: A permissionless distributed ledger that self-adapts to transactions demand," in *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 *Conference on the Theory and Applications of Cryptographic Techniques - Advances in Cryptology (EUROCRYPT)*, 2015.

[11] Y. Sompolinsky and A. Zohar, "Accelerating bitcoin's transaction processing. fast money grows on trees, not chains." [Online]. Available: https://eprint.iacr.org/2013/881.pdf

[12] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "Spectre: A fast and scalable cryptocurrency protocol." [Online]. Available: https://eprint.iacr.org/2016/1159.pdf

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

[14] P.-Y. Piriou and J.-F. Dumas, "Simulation of stochastic blockchain models," in *European Dependable Computing Conference (EDCC)*, 2018.

[15] 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.

[16] E. Rosa, G. D'Angelo, and S. Ferretti, "Agent-based simulation of blockchains." [Online]. Available: https://arxiv.org/pdf/1908.11811.pdf

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

[18] M. Bottone, F. Raimondi, and G. Primiero, "Multi-agent based simulations of block-free distributed ledgers," in *Advanced Information Networking and Applications Workshops (WAINA)*, 2018.

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

[20] U. Wilensky, "Netlogo." [Online]. Available: http://ccl.northwestern.edu/netlogo/

[21] 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

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

[23] G. Bu, Ö. Gürcan, and M. Potop-Butucaru, "G-IOTA: fair and confidence aware tangle." [Online]. Available: https://www.arxiv-vanity.com/papers/1902.09472/

[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 *Workshop on Infrastructure for Multi-Agent Systems*, 2000.

[27] D. Balouek et al., "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science (CLOSER)*, 2013.

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

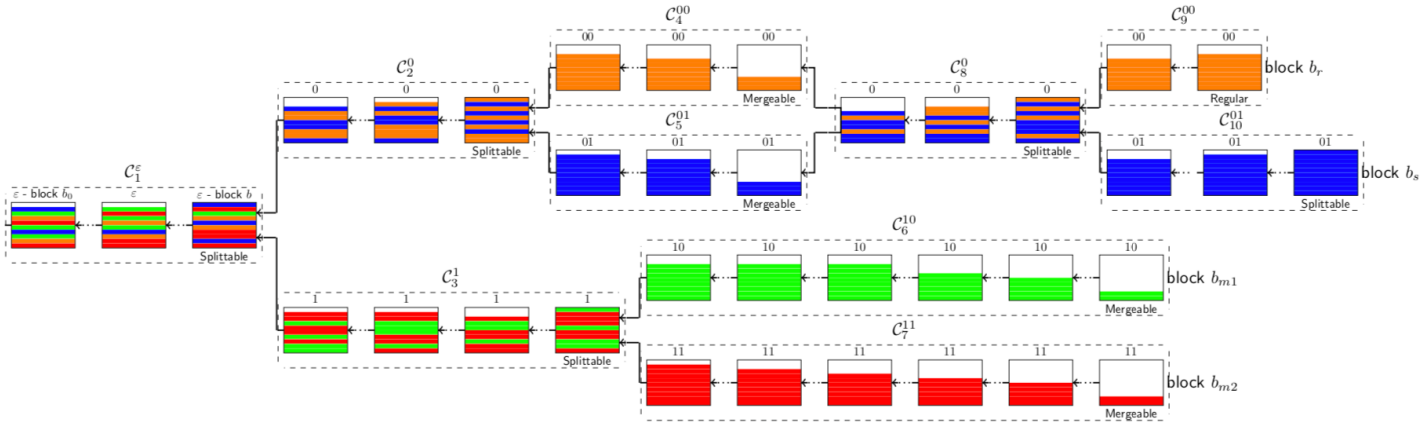


Fig. 4: An example of a SYC-DAG built by Sycomore. This figure has been borrowed from [9]. System parameters: overload threshold $\Gamma = 95\%$, underload threshold $\gamma = 15\%$, and $c_{\min} = 2$.

APPENDIX

Figure 4 illustrates the different notions introduced in Section III. 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. This 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.

Lemma 1 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 total hashrate 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. Let p_{pow} be the success probability of each trial. 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, $P(X = q) = (1 - p_{pow})^{q-1} p_{pow}$. Let W be the total hashrate of the system, and c the current number of leaf chains. The probability p of successfully mining a block is thus given by $p = p_{pow} cW$, and at a miner, this probability is equal to $p_u = p/|\mathcal{U}|$. 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 is equal to $p_c = p/c$. Let X_c be the random variable representing the number of trials before the first success on a given leaf chain, we have $\mathbb{E}(X_c) = 1/p_c = 1/(p_{pow} W)$, and consequently, $\mathbb{E}(X) = 1/p = 1/(p_{pow} cW)$. This completes the proof. □

Lemma 2 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 $\mathcal{C}_i$, $i \in [\![1, c]\!]$, 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. We model the block creation process as a Poisson process. In the following an event represents the creation of a block. The events produced by the Poisson process can be of c different types, c being the number of leaf blocks $b_1^{\ell_1}, \ldots, b_c^{\ell_c}$. An event of type i represents the creation of a block that matches chain $\mathcal{C}_i^{\ell_i}$. 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 $i \in [\![1, c]\!]$. The successive choices for the types are supposed to be independent of each other and also independent of the Poisson process. For every $i \in [\![1, c]\!]$, let $\{N_i(t),\ t \geq 0\}$ be the number of events of type i produced by the Poisson process in the interval $(0, t)$. 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 $(0, t)$ (i.e., a fork occurs in the interval $(0, t)$). We then have, for every $i \in [\![1, c]\!]$, $p_i(t) = \mathrm{P}\{N_i(t) \geq 2\} = 1 - e^{-\lambda t/c}(1 + \lambda t/c)$. Note that this probability holds in Sycomore only at the instants at which periodic readjustments of the difficulty occur. □

Lemma 3 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 valid blocks b_1 and b_2. Two cases must be considered. Case 1: $b_1 = b_2$. 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). Case 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. Sub-case 1: 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. Sub-case 2: 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 1, the ledger with the longest commitment path to the genesis block will be kept by all the nodes, which completes the proof. □