HAL
open science

Handling Bitcoin Conflicts Through a Glimpse of Structure

Thibaut Lajoie-Mazenc, Romaric Ludinard, Emmanuelle Anceaume

To cite this version:

Thibaut Lajoie-Mazenc, Romaric Ludinard, Emmanuelle Anceaume. Handling Bitcoin Conflicts Through a Glimpse of Structure. Proceedings of the 32nd ACM SIGAPP Symposium On Applied Computing, Apr 2017, Marrakesh, Morocco. 10.1145/3019612.3019657 . hal-01634368

HAL Id: hal-01634368

https://hal.science/hal-01634368v1

Submitted on 14 Nov 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Handling Bitcoin Conflicts Through a Glimpse of Structure

Thibaut Lajoie-Mazenc
CNRS, UMR 6074
France
tlm@kth.se

Romaric Ludinard
ENSAI, UMR 9194
France
romaric.ludinard@ensail.fr

Emmanuelle Anceaume
CNRS, UMR 6074
France
anceaume@irisa.fr

ABSTRACT

Double spending and blockchain forks are two main issues that the Bitcoin crypto-system is confronted with. The former refers to an adversary's ability to use the very same coin more than once while the latter reflects the occurrence of transient inconsistencies in the history of the blockchain distributed data structure. We present a new approach to tackle these issues: it consists in adding some local synchronization constraints on Bitcoin's validation operations, and in making these constraints independent from the native blockchain protocol. Synchronization constraints are handled by nodes which are randomly and dynamically chosen in the Bitcoin system. We show that with such an approach, content of the blockchain is consistent with all validated transactions and blocks which guarantees the absence of both double-spending attacks and blockchain forks.

CCS CONCEPTS

•Theory of computation → Probabilistic computation; Design and analysis of algorithms;

KEYWORDS

Bitcoin; Safety; Double-spending attack; Blockchain fork

1 INTRODUCTION

In 2008, Satoshi Nakamoto proposed a scheme for a financial system that does not rely on a central banking authority to ensure its safety: the *Bitcoin cryptocurrency system*, the first decentralized ecosystem providing users with a virtual currency [22]. Bitcoin relies on a public distributed ledger, the so-called *blockchain*, that is an ordered sequence of blocks. Each block confirms a set of valid and recently issued transactions. Since any participant can verify the validity of the entire blockchain, that is the set of all transactions ever confirmed in the system, it does not require them to trust each other: each one is assumed to act selfishly. Thus, the blockchain can be viewed as a way to create a global trusted third-party from a network of untrusted entities. However, the presence of the blockchain does not heal Bitcoin from two important design issues: double-spending attacks and blockchain forks. A *double-spending attack* consists for a malicious user in issuing transactions that use the very same coin to buy goods from different sellers. Such an attack is successful if at least one of the sellers provides his goods and the transaction he is recipient of is not inserted in the blockchain [12]. A *blockchain fork* occurs when concurrent blocks are created, forcing the blockchain to be split in several branches. Even if Bitcoin eventually converges to a state with a unique branch, stabilization may take time. For instance, the March 2013 fork [12] was only resolved after several hours. During a fork, an attacker may repeatedly perform double-spending attacks, inserting conflicting transactions in the conflicting views of the blockchain to make them appear as validated by the network. Given the high value of bitcoins, reported to be worth from \$540 to \$710 between July 1st and August 31st, 2016 [1], one may expect that the abuse of the weaknesses of Bitcoin's design will increase along with its adoption. The root cause of Bitcoin's vulnerability to such attacks is the absence of consistency between Bitcoin operations. Three recent works [9, 11, 18] have proposed to rely exclusively on miners to take in charge the full process of validation and confirmation to guarantee that all the operations triggered on the transactions and on the blockchain are atomically consistent. Beyond the complexity introduced by these approaches, all important decisions of Bitcoin are put solely under the responsibility of (a quorum of) miners, and the membership of the quorum is decided by the quorum members, which magnifies the power of malicious miners [3].

Our contributions In this paper, we show that what fundamentally needs to be done to prevent double-spending attacks and blockchain forks is to impose synchronization constraints on the objects to be validated. In particular, synchronization must guarantee that the request for the validation of the input of a transaction is granted only if it has never been granted before. Similarly, the request for the validation of a block must be granted only if the block that precedes that block has never granted such a request before. We encapsulate minimal synchronization constraints in two conflict detection services, respectively dedicated to transactions and blocks. Their specifications are given in terms of safety and liveness properties. We then describe a simple implementation of both services by enriching the Bitcoin peer-to-peer overlay network with a graph structure.

The remainder of the paper is organized as follows. Section **??** briefly presents the main features of Bitcoin, and Section **??** describes the computational and system model adopted in this work. Section 2 presents the conflict detection services, and Section 3 exhibits a possible implementation of

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SAC 2017,
© 2017 ACM. 978-1-4503-4486-9/17/04. . . \$15.00
DOI: 10.1145/3019612.3019657

the network overlay it requires. A brief survey of some of the attempts that have been made at solving Bitcoin's issues is presented in Section 4. Finally, Section 5 concludes.

2 BACKGROUND ON THE BITCOIN NETWORK

The Bitcoin network [22] is a peer-to-peer payment network that relies on distributed algorithms and cryptographic functions to allow entities to pseudonymously buy goods with digital currencies called bitcoins. Bitcoin mainly relies on two types of objects: transactions and blocks. A transaction consists in two sets: the inputs, the coins being spent, and the outputs, the accounts to which these coins are sent. A block contains a list of transactions, a reference to its parent block (hence the block*chain*), and a *proof-of-work*, that is a nonce such that the hash of the block matches a given target. This target is calibrated so that the mean generation time of a block is equal to 10 minutes. Transactions are issued by the *users* of the system and are propagated by the *Bitcoin nodes* (*i.e.*, the nodes participating to the Bitcoin peer-to-peer network), as soon as they have been locally validated. Informally, a transaction $T = (I, O)$ is locally valid at node p if p has received all the transactions that have credited all the inputs in I and has never received transactions already using any of those inputs. If there exists some transaction $T' = (I', O')$ and some input i such that $i \in I \cap I' \neq \emptyset$, then input i is said to be in a *double-spending situation*. Transaction $T = (I, O)$ is *conflict-free* if none of the inputs of T is involved in a double-spending situation and all of the transactions that credited T's inputs are conflict-free. By construction, the induction is finite because Bitcoin creates money only through coinbase transactions, which are by definition conflict-free [3]. Blocks are generated by *miners*, a subset of the nodes involved in the *proof-of-work* competition. The incentive to participate to such a competition is provided by a reward given to each successful miner, this reward consisting of a fixed amount of coins (currently 12.5 bitcoins) and a fee associated to each transaction contained in the newly generated block. When a transaction T is included in a block b, it is said *confirmed* by all the peers that accept that block in their local copy of the blockchain. The *level of confirmation* of transaction T is the number of blocks included in the blockchain starting from b; by extension, a 0 confirmation level means that the transaction has not yet been included in the blockchain. To limit double-spending attacks, Bitcoin recommends that sellers do not provide their goods in exchange of a transaction before it becomes *deeply-confirmed*. Actually, Nakamoto [22] as well as subsequent studies [13, 17, 21] have shown that if the proportion of malicious miners μ is equal to 10%, then with probability less than 0.1%, a transaction can be rejected if its level of confirmation in a local copy of the blockchain is less than 5. The level of confirmation must increase to 8 when μ increases to $\mu = 15\%$, and to 15 when 25% of the miners are corrupted. In the following, we say that a transaction is *deeply confirmed* once it reaches such a confirmation level. Now in case of a blockchain fork, some blocks can be invalidated and the level of confirmation of their transactions can decrease, especially if the conflicting branch contains a conflicting transaction. This deters the use of Bitcoin for fast payment, as the expected time for a deep confirmation is approximately one hour. Fast payment are used in most everyday life situations, where the time between buying and consuming the goods is in the order of minutes. This impracticality motivates this work.

3 MODEL

In this section we formalize the assumptions related to the Bitcoin system. Specifically, we assume a large, finite yet unbounded set Π of nodes whose composition may change over time. Each node of Π has identical networking and computational capabilities. The communication delays between any two nodes, the time to execute a local computation step, and the drift of local clocks are assumed to be upper-bounded, however these upper-bounds are unknown to nodes. These temporal assumptions fit the partial synchrony model [10]. The Bitcoin ecosystem being an open and dynamic system, the presence of malicious behaviors is unavoidable. In this paper, we assume that a bounded proportion μ, with $\mu \leq \lfloor 1/(3+\varepsilon) \rfloor$ for some $\varepsilon > 0$, of the nodes in Π are malicious or Byzantine, that is can deviate arbitrarily far from the specified protocol.[1] Nodes do not have access to any trusted PKI infrastructure to establish their identities. Thus, each node must have the possibility to create its own identities. As will be detailed later, to prevent malicious nodes from generating large set of identities to simulate many different nodes, each node needs to solve computationally expensive proof-of-works to create their identities. The cryptographic primitives used in Bitcoin (hash functions and digital signatures) are assumed to be safe (forging signatures, and finding hash collisions or pre-image are all provably impossible for our computationally bounded nodes). We suppose that objects (*i.e.*, inputs, transactions and blocks) have unique IDs distributed uniformly at random over a finite subset of $\mathbb{N}$. The ID of object x is denoted by $h(x)$ regardless of the type of x, where h represents Bitcoin's 256-bit hash (corresponding to applying twice SHA-256 on the input). We justify the assumptions about object IDs by the fact that Bitcoin already considers the double SHA-256 hash of blocks and transactions to uniquely identify them. We extend it to inputs by considering the double SHA-256 hash of the pointer to the coin being spent (from an implementation point of view, the hash of the transaction that last spent it and the index of the output to which it was sent). The uniform distribution holds in practice, as shown in figure **??**.

[1] Note that as we are focusing on a financial cryptosystem, we cannot just consider that nodes are either obedient (*i.e.*, they follow the prescribed algorithm) or malicious. Most of the nodes are rational that is, strategically behave to increase their own utility function without violating the prescribed protocols. For example, a rational miner may in priority insert in its block all the current transactions that provide the maximal fees while an obedient one will insert transactions irrespective of the gain they procure. Since rational nodes do not violate the protocol specification, in the following we consider as correct both obedient and rational nodes.

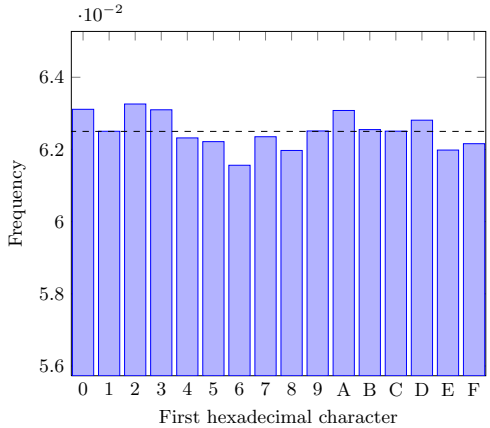

Figure 1: Distribution of transactions and inputs.

This figure depicts the frequency at which each hexadecimal character appears as the first hexadecimal character of $h(x)$, for x iterating over the set of transactions contained in 100 consecutive blocks starting at height 420,000 and all of their inputs. Note that 19 transactions were excluded because they were too big to be decoded by Bitcoin Core's RPC API; thus, this study covers 368,327 hash results over 102,283 transactions. The dashed line represents the mean, equal to 0.0625 as expected from a uniform distribution over 16 values. The low standard deviation of $4.78e^{-4}$ (with Bessel's correction) confirms the good performances of the hash. Results are similar when considering the last hexadecimal character instead ($0.0625 \pm 4.22e^{-4}$). Finally, we assume that all the objects (*i.e.*, inputs, transactions and blocks) are well-formed. In particular a transaction can be rejected (by a correct node) only if that transaction tries to double-spend inputs and not *e.g.* because the input scripts are incorrect. In addition, we suppose that all transactions contain transaction fees, and those fees are sufficient to incentivize any miner to include all the received transactions in their blocks.

4 CONFLICT DETECTION SERVICES

The present work aims at understanding which is the least amount of synchronization that needs to be introduced in the Bitcoin ecosystem to prevent double spending attacks and blockchain forks through consistent conflict resolutions for both transactions and blocks. As will be detailed in the following, we encapsulate all the synchronization constraints into two services, that we call *transaction conflict detection service* and *block conflict detection service*, respectfully dedicated to transaction and block synchronizations. Prior to presenting the specification of both services, we first recall the properties currently met by the Bitcoin ecosystem, properties specified in terms of liveness and safety properties [3]:

Safety: If a transaction T is deeply confirmed by some correct node, then no transaction conflicting with T will ever be deeply confirmed by any correct node.

Liveness: A conflict-free transaction will eventually be deeply confirmed in the blockchain of all correct nodes at the same height in the blockchain.

In the present paper, by preventing double-spending attacks and blockchain forks we aim at strengthening Bitcoin's safety property as follows:

Strong Safety: If a transaction T is confirmed by some correct node, then no transaction conflicting with T will ever be confirmed by any correct node.

The strong safety property ensures that whenever a transaction T has been included in a block, no other block will ever contain a transaction conflicting with T. An immediate and important consequence of this property is the capability for Bitcoin to safely handle fast payments. The remaining of the paper is devoted to the implementation of this property.

In the following, notations $T = (I, O)$ and $b' = (h(b), c(b'))$ respectively denote the transaction T with input set I and output set O, and the block b' built on top of block b and containing the transactions $c(b')$.

4.1 Specification of the CDSs

4.1.1 Transaction Conflict Detection Service. The purpose of the transaction conflict detection service (TCDS) is to ensure that concurrent transactions do not try to use common inputs. If we make an analogy between *transaction inputs* and *objects*, and an analogy between *using an input* and *writing an object*, then we can refer to database systems, in which exclusive access to objects is obtained by asking each transaction to explicitly lock objects it accesses using some single object locking mechanism. Yet, unless care is taken, locking objects one by one may cause deadlocks. As the application we consider involves different entities spread over a large area, it is not advisable to rely on having all of them conform to the same locking strategies. Moreover, from a performance viewpoint, it may be impossible to run deadlock detection and prevention protocols assuming independent object locking. In the following we propose a TCDS that provides the equivalent of an atomic locking mechanism for all of the inputs of each issued transaction. Formally, the TCDS offers two methods, the `grantInputs` method and the `Release` one, that both accept a transaction $T = (I, O)$ as parameter. The `grantInputs` method returns with GRANTED or DENIED. When an invocation returns with GRANTED, we say that *the method exclusively grants the inputs in I to T* or, in short, that *T has been* GRANTED. We say that T *releases* objects previously granted to it either when T invokes the `Release` method, or when all the nodes that participate in the TCDS for T have crashed or become permanently disconnected. In the former case, T releases all its locks. Based on this definition, we require the TCDS to provide the following properties:

Safety: If a transaction $T = (I, O)$ is exclusively granted the inputs in I, then no other transaction $T' = (I', O')$ is exclusively granted the inputs in I' with $I \cap I' \neq \emptyset$.

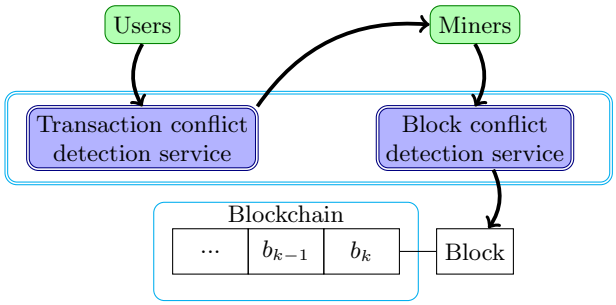

Figure 2: Orchestration of the CDS services.

Liveness: Each invocation of the `grantInputs` method eventually returns.

Non triviality: If there exists an invocation of the `grantInputs` method with $T = (I, O)$, and no other transaction $T' = (I', O')$ with $I \cap I' \neq \emptyset$ is exclusively granted the inputs in I' then T is granted exclusively all the inputs in I.

4.1.2 Block Conflict Detection Service. The block conflict detection service (BCDS) aims at ensuring that any validated block has at most one valid block as immediate successor. It offers a single method, `grantBlock`, that accepts a block $b' = (h(b), c(b'))$ as parameter. This method returns with GRANTED or DENIED. When an invocation returns with GRANTED, we say that *the method validates* block b' as the unique successor of block b. Based on this definition, we require the block conflict detector to provide the following properties:

Safety: If a block $b' = (h(b), c(b'))$ is granted $h(b)$, then no other block $b'' = (h(b), c(b''))$ is granted $h(b)$.

Liveness: Each invocation of the `grantBlock` method eventually returns.

Non triviality: If there exists an invocation of the `grantBlock` method with $b' = (h(b), c(b'))$, and no invocation of `grantBlock` with $b'' = (h(b), c(b''))$ has ever been granted, then block b' is granted as the unique successor of block b.

Transactions are *confirmed* once they are included in a block that has been granted by the BCDS. Figure 1 depicts the path of transactions from the users to the blockchain.

4.2 Orchestration of the CDSs

In order to implement the Conflict Detector Services we introduce the notion of referees. The main idea is to associate a randomly chosen node of the Bitcoin system to each object o (i.e., input, transaction or block) in charge of handling all the CDS operations on object o. This specific node is called the referee pi_o of o. In the remaining of this paragraph we assume that for each object o, its referee pi_o is known. Section 3 will provide a simple way to determine the referee of each object.

When a user creates a transaction $T = (I, O)$, it sends T to its referee pi_T so that pi_T can invoke TCDS with T. This invocation consists for pi_T in asking an exclusive lock at the referee pi_i of each input $i \in I$, in an order that corresponds to the lexicographical order of the input IDs. If the lock is DENIED for at least one of these inputs, pi_T releases all previously obtained locks (by proving to each of these referees that a conflicting transaction T' has already been GRANTED). Otherwise, after obtaining all locks, a GRANTED status is returned to pi_T.

The correctness and, in particular, the lack of deadlocks, result from the fact that objects are always obtained in lexicographical order. A lock can be implemented using a combination of Test-and-Set and Reset primitives. The referee pi_i that wishes to lock input $i \in I$, first checks the value of a binary register. When this value is 0, it modifies the register to 1 and uses the lock. Releasing a lock is done by resetting to 0 the register value. The fact that T has been granted the lock on each input $i \in I$ is proven by pi_i's signature. Each signature is bundled with the identity of the signer. Note that Bitcoin transactions can easily be extended to accommodate this process: the referee pi_T of transaction $T = (I, O)$ computes a group signature S (*e.g.* [7]) using the signatures of each input referee $pi_{i_{i \in I}}$ and its own signature and appends it, along with everything needed to verify this group signature, to a specific *validation* output o added to the set of outputs O of transaction $T = (I, O)$. Any node can easily verify that transaction $T = (I, O)$ has been GRANTED by checking the signatures S added by referee pi_T.

Referees are incentivized by introducing a *validation fee*. A fair and easy way to share the *validation* output is to randomly pick one of the referees and give it the entire reward. This requires seeding a random number generator in a publicly verifiable way, and for example with an information that can only be published after the TCDS has returned like the hash of the block in which the transaction is included.

The process is even simpler for blocks: when a miner creates a block $b' = (h(b), c(b'))$, it sends b' to its referee $pi_{b'}$ so that $pi_{b'}$ can invoke the BCDS with b'. This invocation consists for $pi_{b'}$ in asking an exclusive lock at referee pi_b. If the lock is GRANTED, then both pi_b and $pi_{b'}$ signatures are applied to the *coinbase* of b', that is the transaction that gives the miner the reward for generating b'. Note that the hash used in the proof of work does not cover the referee signatures.

5 LEVERAGING DHTS TO IMPLEMENT THE CDSS

The fundamental principle of the above two CDSs is the link between each Bitcoin object (*i.e.*, transaction, input, and block) and its referee. By doing so, each input is granted an exclusive access to a unique transaction, and each block has at any time at most one unique successor. The solution we propose to implement such a link simply consists in bringing some structure to the underlying unstructured peer-to-peer overlay of Bitcoin. Recall that the topology of unstructured

overlays conforms random graphs, i.e., relationships among nodes are mostly set according to a random process, and routing is not constrained. Object placement enjoy the same absence of constraints. Any data can be placed on any node thereby imposing flooding or random walk techniques to retrieve them. On the other hand, structured overlays, also called Distributed Hash Tables (DHTs), build their topology according to structured graphs. For most of them, the following principles hold: the identifier space is partitioned among all the nodes of the overlay. Nodes self-organize within the graph according to a distance function D based on node IDs (*e.g.* two nodes are neighbors if their IDs share some common prefix), plus possibly other criteria such as geographical distance. Each application-specific object is assigned a unique identifier selected from the same identifier space. Each node owns a fraction of all the objects of the system. The mapping derives from the distance function D.

Any DHT could be a valuable candidate to organize nodes and objects in Bitcoin, as long as the chosen DHT is capable of handling high churn [15] and the presence of colluding Byzantine nodes. Among the possible candidates, the hypercube PeerCube [4] enjoys both properties. Briefly, PeerCube gathers nodes into clusters, so that each vertex of the hypercube is composed of a cluster of nodes. All the routing and storage operations classically devoted to each single node in a non-clustered DHT are jointly handled by a small and constant size subset of the nodes of each single cluster through Byzantine-tolerant consensus protocols. In addition, the impact of churn is handled at cluster level which minimizes the impact on the graph structure of the DHT. Finally, PeerCube limits the sojourn time of nodes at the same position of the overlay (through induced churn [5, 6] to prevent the adversary from choosing its own positions and from eclipsing correct nodes from a given region of the overlay. Coming back to the implementation of CDSs, a simple way to determine the referee pi_o of each object o is to choose the closest vertex (i.e. the closest cluster of nodes) to $h(o)$, which guarantees the robustness of this referee.

Despite these enjoyable qualities, PeerCube, and DHTs in general, assume the presence of a trusted third-party to act as a public key infrastructure (PKI) in charge of assigning certified identities to each node. It turns out that such an assumption is unrealistic in large scale and dynamic systems, and consequently, we can only rely on nodes to create themselves their identities. Since up to a proportion μ of the nodes are Byzantine, there is a risk of Sybil attacks (*i.e*, the creation of a considerable number of identities) if it is profitable to own several identities. To drastically limit the number of identities per node, we propose to leverage the *proof of work* mechanism of Bitcoin: each node must solve a computationally expensive challenge to create an identity [20], which in expectation makes the number of identities per node proportional to its computational power. We propose to build an identity $\mathcal{I}$ as follows: identity $\mathcal{I}$ should comprise a public key $\mathrm{PK}_{\mathcal{I}}$, a timestamp $t_{\mathcal{I}}$, a nonce $v_{\mathcal{I}}$, and the hash of the last known block $h(b_{\mathcal{I}})$ of the blockchain. The public key authenticates messages, while the timestamp forces an induced churn: identities have a lifetime that periodically expires which allows correct nodes to escape poisoning attacks by moving to a new region in the overall, and by preventing malicious nodes from staying indefinitely long in the same region of the overlay. Hence, node $\mathcal{I}$ recomputes a new identity $\mathcal{I}'$ when Δ blocks are appended to block $b_{\mathcal{I}}$, the block at which node $\mathcal{I}$ created its identity I. Finally, assuming that $\mathcal{I}$ has not expired yet, an identity is considered valid if and only if $h(\mathrm{PK}_{\mathcal{I}}||t_{\mathcal{I}}||v_{\mathcal{I}}||h(b_{\mathcal{I}})) < \mathcal{T}$, where $\mathcal{T}$ is a network-specified target, and $||$ represents the concatenation operator.

Remark that there is no strict guarantee on the actual number of identities a given node from Π controls. This explains why we only require that the proportion μ of Byzantine nodes verifies $\mu \leq 1/(3+\varepsilon)$ for some $\varepsilon > 0$. A precise analysis is left for future work.

Remark 1. *A interesting side-effect of guaranteeing that the k-th block is validated only if it is the first (and thus the unique) successor of the $(k-1)$-th one is that it defends the Bitcoin ecosystem from other forms of adversarial mining. The first one, called by the Bitcoin community the SPV mining (for Simplified Payment Verification), consists for a miner in creating a block, say the k-th one, by only waiting for the hash of the $(k-1)$-th one to be published in the system, rather than, as prescribed by the protocol, by waiting for the block in its entirety to be disseminated. By doing so, a miner starts the mining process without being penalized by the time it needs for the full block to be propagated in the system. The second form of adversarial mining, the selfish mining, consists for a miner in trying to eclipse the last blocks of the blockchain with its own sequence of blocks. This is achievable if the miner has very large computational resources, and succeeds in creating a sequence of blocks whose size is larger than the current one in the blockchain. Both undesirable adversarial mining attacks are impeded by locally synchronizing objects: the hash used to refer to a block when mining on top of it covers the referee's signature. Thus, it becomes pointless to keep newly found blocks secret, and mining pools cannot publish the hash of the blocks they find on publicly available servers before they have been accepted by the BCDS.*

6 RELATED WORK

Bitcoin [22] is seen as the pioneer of cryptocurrencies. Since its inception, several altcoins [2] have emerged. The GHOST protocol [23] proposes a different rule to solve blockchain forks, based on the number of blocks contained in each blockchain subtree (in case of consecutive forks). Recent works have focused on Bitcoin modeling and evaluation. Authors of [21] prove that the Bitcoin protocol achieves consensus with high probability, while [13] show that peers participating in the Bitcoin network agree on a common prefix for the transaction history, both in failure-free environments. In contrast, authors of [16, 17] focused on adversarial environments. These works study the feasibility of double spending attacks and their detection. Several studies have shown that Bitcoin behaves quite well in failure-free environments [21] but is vulnerable to some attacks such as the double-spending

one [17]. Several attempts to fix it have been published, using a leader [9, 11, 18], or forming local committees to run consensus algorithms at the local level [20] but these proposals encounter various scalability or security issues which make them unusable. Specifically, Bitcoin-NG [11], PeerCensus [9], and BizCoin [18] have proposed to rely exclusively on miners to take in charge the full process of validation and confirmation to guarantee that all the operations triggered on the transactions are atomically consistent. In all these protocols, time is divided into epochs. An epoch ends when a miner successfully generates a new block. This miner becomes the leader of the subsequent epoch. Strong consistency is implemented in these protocols by different means. In Bitcoin-NG, it is achieved by delegating the validation process to the leader of the current epoch. In PeerCensus it is implemented by relying on Byzantine Fault Tolerant consensus protocols (*e.g.* [8, 14, 19]) run by all the miners that successfully generated a block. Finally, BizCoin leverages both ideas by using the leader and a consensus run by the last w successful miners. In all these protocols, the dedicated miners are entitled to validate and confirm issued transactions and blocks and to disseminate them so that each peer integrates them in its local blockchain. It has been shown in a previous paper [3] that none of the studied solutions enhances Bitcoin's behavior. Beyond the complexity introduced by the consensus executions, the main issue comes from the fact that all important decisions of Bitcoin are solely under the responsibility of (a quorum of) miners, and the membership of the quorum is decided by the quorum members. This magnifies the power of malicious miners.

7 CONCLUSION

To prevent double spending attacks, Nakamoto's analysis, and subsequent ones [13, 17, 21] have shown that users can safely deliver their good once their validated transaction has been confirmed for a sufficiently long time. This fully relies on the intuition that pools of malicious miners have no collective incentive to aggregate their computation power in order to prune the blockchain at almost any position in the blockchain. In this paper we have shown that introducing a small amount of synchronization, soon enough in the validation process, allows a user to deliver his good as soon as the transaction has been validated by the system. Indeed, this new transaction validation scheme ensures that any valid transaction will remain in the blockchain forever, which is a strong guarantee for the user to be paid even for fast payments transactions. The same level of local synchronization heals the Bitcoin system from blockchain forks, SPV mining and selfish minings. We continue our quest to make this distributed structure safe and scalable through a deep analysis of its properties both in terms of safety, liveness and performance.

REFERENCES

[1] 2016. Blockchain.info. https://blockchain.info. (2016). Online, accessed September 9th.

[2] Shaikshakeel Ahamad, Madhusoodhnan Nair, and Biju Varghese. 2013. A Survey on Crypto Currencies. In *Proceedings of the International Conference on Advances in Computer Science (AETACS)*.

[3] Emmanuelle Anceaume, Thibaut Lajoie-Mazenc, Romaric Ludinard, and Bruno Sericola. 2016. Safety Analysis of Bitcoin Improvement Proposals. In *15th IEEE International Symposium on Network Computing and Applications (NCA)*.

[4] Emmanuelle Anceaume, Romaric Ludinard, Aina Ravoaja, and Francisco Brasileiro. 2008. PeerCube: A Hypercube-Based P2P Overlay Robust against Collusion and Churn. In *2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*.

[5] Emmanuelle Anceaume, Romaric Ludinard, and Bruno Sericola. 2012. Performance evaluation of large-scale dynamic systems. *ACM SIGMETRICS Performance Evaluation Review* 39, 4 (2012), 108–117.

[6] B. Awerbuch and C. Scheideler. 2004. Group spreading: A protocol for provably secure distributed name service. In *Proceedings of the 31rst International Colloquium on Au- tomata, Languages and Programming (ICALP)*.

[7] Alexandra Boldyreva. 2003. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *6th International Workshop on Practice and Theory in Public Key Cryptography (2003)*.

[8] M. Castro and B. Liskov. 1999. Practical Byzantine Fault Tolerance. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*.

[9] Christian Decker, Jochen Seidel, and Roger Wattenhofer. 2016. Bitcoin Meets Strong Consistency. In *17th International Conference on Distributed Computing and Networking (ICDCN)*.

[10] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the Presence of Partial Synchrony. *J. ACM* (1988).

[11] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. 2016. Bitcoin-NG: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[12] Neil Fincham. 2013. https://mineforeman.com/2013/03/14/what-the-fork-was-that-a-forking-post-mortem/. (2013).

[13] J. A. Garay, A. Kiayias, and N. Leonardos. 2015. 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)*.

[14] R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić. 2010. The Next 700 BFT Protocols. In *Proceedings of the European Conference on Computer Systems (EuroSys)*.

[15] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In *24th USENIX Security Symposium*.

[16] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. 2012. Double-spending Fast Payments in Bitcoin. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS)*.

[17] Ghassan O. Karame, Elli Androulaki, Marc Roeschlin, Arthur Gervais, and Srdjan Čapkun. 2015. Misbehavior in Bitcoin: A Study of Double-Spending and Accountability. *ACM Trans. Inf. Syst. Secur.* 18, 1 (2015).

[18] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *25th USENIX Security Symposium*.

[19] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. 2007. Zyzzyva: Speculative byzantine fault tolerance. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*.

[20] Loi Luu, Viswesh Narayanan, Kunal Baweja, Chaodong Zheng, Seth Gilbert, and Prateek Saxena. 2015. *SCP: a computationally-scalable Byzantine consensus protocol for blockchains*. Technical Report. Cryptology ePrint Archive, Report 2015/1168.

[21] Andrew Miller and Joseph J LaViola Jr. 2014. Anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin. http://bravenewcoin.com/assets/Whitepapers/. (2014).

[22] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[23] Y. Sompolinsky and A. Zohar. 2013. Accelerating Bitcoin's Transaction Processing. Fast Money Grows on Trees, Not Chains. *IACR Cryptology ePrint Archive* 2013 (2013), 881.