ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/319984012

From blockchain consensus back to Byzantine consensus

Article *in* Future Generation Computer Systems · September 2017

DOI: 10.1016/j.future.2017.09.023

CITATIONS
9

READS
488

1 author:

Vincent Gramoli
Data61-CSIRO and University of Sydney

126 PUBLICATIONS **996** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project The Red Belly Blockchain View project

Project Data Structures for Multicores View project

All content following this page was uploaded by Vincent Gramoli on 01 September 2018.

The user has requested enhancement of the downloaded file.

From Blockchain Consensus Back to Byzantine Consensus

Vincent Gramoli
Data61-CSIRO and University of Sydney
Australia

Abstract

Consensus is a fundamental problem of distributed computing. While this problem has been known to be unsolvable since 1985, existing protocols were designed these past three decades to solve consensus under various assumptions. Today, with the recent advent of blockchains, various consensus implementations were proposed to make replicas reach an agreement on the order of transactions updating what is often referred to as a distributed ledger. Very little work has however been devoted to explore its theoretical ramifications. As a result existing proposals are sometimes misunderstood and it is often unclear whether the problems arising during their executions are due to implementation bugs or more fundamental design issues.

In this paper, we discuss the mainstream blockchain consensus algorithms and how the classic Byzantine consensus can be revisited for the blockchain context. In particular, we discuss proof-of-work consensus and illustrate the differences between the Bitcoin and the Ethereum proof-of-work consensus algorithms. Based on these definitions, we warn about the dangers of using these blockchains without understanding precisely the guarantees their consensus algorithm offers. In particular, we survey attacks against the Bitcoin and the Ethereum consensus algorithms. We finally discuss the advantage of the recent Blockchain Byzantine consensus definition over previous definitions, and the promises offered by emerging consistent blockchains.

1 Introduction

The blockchain technology [33] promises to radically transform the way individuals and companies exchange digital assets and track securely ownership of these assets without the control of a central authority. At its heart lies a distributed ledger that is consistent with high probability when particular assumptions are fulfilled. In particular, the distributed set of participants guarantee its consistency despite potentially malicious participants that behave arbitrarily, also called *Byzantine* failures [29].

The novelty of blockchain is a genuine combination of well-known research results taken from distributed computing, cryptography and game theory. Its distributed nature guarantees the persistence of the ledger data. Its public key crypto-system offers the capabilities for a user to sign transactions that transfer assets from her account to other accounts. Its incentive mechanisms guarantee that a subset of participants maintain the validity of the transactions. And finally, a Byzantine tolerant consensus protocol aims at guaranteeing the integrity of the ledgers by defining a total order on newly appended blocks of transactions.

Put into the blockchain context, the consensus problem is for the non-faulty or *correct* processes of a distributed system to agree on one block of transaction at a given index of a chain of block. This consensus problem can be stated along three properties: (i) agreement: no two correct processes decided different blocks; (ii) validity: the decided block is a block that was proposed by one process;

(iii) termination: all correct processes eventually decide. A protocol solving the consensus problem is necessary to guarantee that blocks are totally ordered, hence preventing concurrently appended blocks from containing conflicting transactions.

Today, with the recent advent of blockchains, various consensus implementations were proposed to make replicas reach an agreement on the order of blocks of transactions updating the distributed ledger. However, consensus has been known to be unsolvable since 1985. While existing protocols were designed these past three decades to solve consensus under various assumptions, it remains unclear what are the guarantees offered by blockchain consensus algorithms and what are the necessary conditions for these guarantees to be satisfied. While the source code of most blockchain protocols is publicly available, the theoretical ramifications of the blockchain abstraction are rather informal. As main blockchain systems, like Bitcoin [33] and Ethereum [46], are now used to trade millions of US$ every day[1], it has become crucial to precisely identified its theoretical ramifications to anticipate the situations where large volume of assets could be lost.

In this paper, we illustrate the danger of using proof-of-work blockchain without understanding precisely their guarantees by listing vulnerabilities that affect the predominant proof-of-work blockchain systems, namely Bitcoin and Ethereum.[2] To this end, we describe the consensus algorithms at the heart of these two blockchain systems. We also relate these consensus algorithms to decades of research on the topic of distributed computing. More precisely, we identify situations where proof-of-work blockchain consensus is violated by: (i) presenting a survey of existing attacks against the Bitcoin consensus protocol and (ii) showing how Ethereum, which copes with some of these attacks, may suffer from recent attacks, namely the blockchain anomaly [35] and the balance attack [34]. We elaborate on the risks for users to misconfigure proof-of-work blockchain systems when deploying them as a private and consortium blockchains and our own experience with the settings of the R3 Ethereum testbed. The fact that both main proof-of-work blockchains are vulnerable allows us to conclude that more research is necessary to design safe consensus algorithms suited for blockchains.

The rest of the paper is organized as follows. Section 2 presents the general blockchain model. Section 3 introduces the classic Byzantine consensus problem and the probabilistic variant of it. Section 4 specifies the differences of the consensus algorithms used in Bitcoin and Ethereum. Section 5 describes the attacks against Bitcoin and two recent attacks against the Ethereum consensus algorithm. Section 6 redefines the Byzantine consensus in the light of the blockchain context. Section 7 discusses the consortium model and recent reliable consensus proposals. Section 8 concludes.

2 The General Proof-Of-Work Blockchain Model

In this section we model a simple distributed system as a communication graph that implements a blockchain abstraction as a directed acyclic graph. We propose a high-level pseudocode representation of proof-of-work blockchain protocols in this model.

2.1 A simple distributed model for blockchains

We consider a communication graph $G = \langle V, E \rangle$ with processes V connected to each other through fixed communication links E. Processes are part of a blockchain system S. In this paper, we only consider proof-of-work blockchain systems and focus our attention to Bitcoin and Ethereum. Processes can act as clients by issuing transactions to the system and/or servers by *mining*, the action of trying to combine transactions into a block as described in Section 2.2. For the sake of simplicity, we consider that each process possesses a single account (or *address*) and that a *transaction* issued by process p_i is a

[1] https://coinmarketcap.com/exchanges/volume/24-hour/.

[2] Bitcoin and Ethereum are the current most important blockchain systems in terms of market capitalisation.

transfer of digital assets or *coins* from the account of the source process p_i to the account of a destination process $p_j \neq p_i$. Each transaction is uniquely identified and broadcast to all processes in a best-effort manner. We assume that a process re-issuing the same transfer multiple times creates as many distinct transactions.

Processes that initiate the consensus protocol are called *miners*, they initiate the consensus through a propose function depicted at lines 7–12 of Alg. 1 allowing them to propose new blocks. Processes decide upon a new block at a given index at line 18 depending on a function get-main-branch that is specific to the type of proof-of-work blockchain system in use (cf. Sections 4.3.1 and 4.3.2 for Bitcoin and Ethereum corresponding function, respectively). We refer to the computational power of a miner as its *mining power* and we denote the total mining power t as the sum of the mining powers of all miners in V. Each miner tries to group a set T of transactions it heard about into a block $b \supseteq T$ as long as transactions of T do not conflict and that the account balances remain non-negative. For the sake of simplicity in the presentation, the graph G is static meaning that no processes can join and leave the system, however, processes may fail as described in Section 2.3.

Algorithm 1 The general proof-of-work blockchain consensus algorithm at process p_i

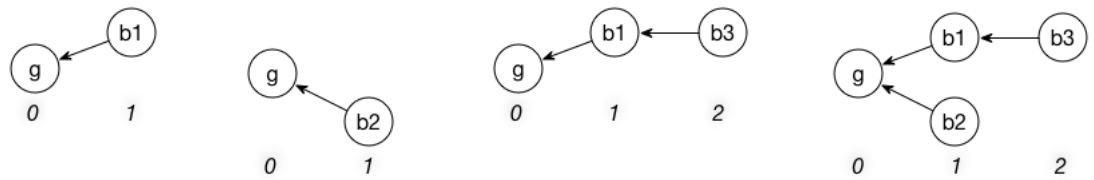
```
1:  ℓ_i = ⟨B_i, P_i⟩, the local blockchain at node p_i is a directed acyclic
2:     graph of blocks B_i and pointers P_i
3:  b, a block record with fields:
4:     parent, the block preceding b in the chain, initially ⊥
5:     pow, the proof-of-work nounce of b that solves the cryptopuzzle, initially ⊥
6:     children, the successor blocks of b in the chain

7:  propose()_i:                                          ▷ function invoked to solve consensus
8:     while true do                                      ▷ do forever
9:        nounce = local-random-coin()                    ▷ toss a local coin to get a nounce
10:       create block b : b.parent = last-block(ℓ_i) and b.pow = nounce   ▷ create a new block
11:       if solve-cryptopuzzle(nounce, b) then           ▷ if the chosen nounce solves the puzzle
12:          broadcast(⟨{b}, {⟨b, b.parent⟩}⟩)            ▷ broadcast to all (including to himself)

13: deliver(⟨B_j, P_j⟩)_i:                                ▷ upon reception of blocks
14:    B_i ← B_i ∪ B_j                                    ▷ update vertices of blockchain
15:    P_i ← P_i ∪ P_j                                    ▷ update edges of blockchain
16:    ⟨B'_i, P'_i⟩ ← get-main-branch()                   ▷ recompute the main branch
17:    if b_0 ∈ B'_i ∧ ∃b_1, ..., b_m ∈ B_i : ⟨b_1, b_0⟩, ⟨b_2, b_1⟩..., ⟨b_m, b_{m−1}⟩ ∈ P_i then   ▷ if enough blocks
18:       decide(b_0)                                     ▷ consensus is reached
```

2.2 Miners must solve a cryptopuzzle to create a new block

Miners provably solve a hashcash cryptopuzzle [9] before creating a new block. Given a global threshold and the block of largest index the miner knows (Alg. 1, line 10), the miner repeatedly selects a nonce and applies a pseudo-random function to this block and the selected nonce until it obtains a result lower than the threshold, this mechanism is hidden within the solve-cryptopuzzle function at line 11 for clarity. Upon success the miner creates a block that contains the successful nounce as a proof-of-work as well as the hash of the previous block, hence fixing the index of the block, and broadcasts the block (line 12). This broadcast function with some block and pointer parameter at process p_i triggers a corresponding deliver function with the same parameter that is invoked at each correct process p_j (including p_i if it is correct) upon reception of this broadcast message (line 13). As there is no known strategy to solve the cryptopuzzle, the miners simply keep testing whether randomly chosen numbers solve the cryptopuzzle with brute force. The *difficulty* of this cryptopuzzle, defined by the threshold, limits the rate at which new blocks can be generated by the network.



(a) view ℓ_1 (b) view ℓ_2 (c) view ℓ_3 (d) global state $\ell_0 = \ell_1 \cup \ell_2 \cup \ell_3$

Figure 1: The global state ℓ_0 of a blockchain results from the union of the distributed local views ℓ_1, ℓ_2 and ℓ_3 of the blockchain

2.3 The failure model

The value of coins in blockchain systems incentivizes participants to act maliciously if they can maximize their gain by for example executing a *double-spending* attack, spending the same coins in two distinct transactions. Malicious behaviors are generally modeled by an arbitrary or Byzantine failure model that is named after the problem of generals attempting to reach an agreement in the presence of traitors [29]. In the Byzantine failure model, such a problem is usually referred to as the *Byzantine consensus* problem. Some solutions like PBFT [13] were proposed, however, there are known not to scale to a number of participants as large as mainstream public blockchain systems [16, 44].

There are different failure models that separate *faulty* processes from *correct* processes. In a crash failure model, the faulty processes may crash at which point they stop computing and stop communicating for the rest of the execution. In a Byzantine failure model, the faulty processes may not follow the protocol specification by behaving arbitrarily. When not stated otherwise, we consider a Byzantine failure model and assume the presence of an *adversary* (or attacker) that can control processes that together own a relatively small fraction $\rho < 0.5$ of the total mining power of the system. The processes controlled by the adversary are called *malicious* or Byzantine and may not follow the protocol specification, however, they cannot impersonate other processes while issuing transactions.[3] We also assume that the adversary can transiently disrupt communications on a selected subset of edges E_0 of the communication graph G.

3 The Consensus Problem for the General Model

Blockchain systems resemble replicated state machine [47] and aim at solving the consensus problem, so that for a given index all correct processes agree on a unique block of transactions at this index. Note that nodes may propose different blocks at the same index because remote miners solve cryptopuzzles in the time it takes to exchange their new resulting block—this is generally observed with a fork as we will explain in Section 4.2. The classic definition of consensus in the Byzantine failure model is either called Byzantine agreement or *Byzantine consensus* and is defined along three properties.

Definition 1 (Byzantine Consensus)**.** *The* Byzantine Consensus *problem is to guarantee the conjunction of these three properties for a given index:*

[3]This is typically ensured through public key crypto-systems.

- **Agreement**: *no two correct processes decide different blocks;*
- **Termination**: *all correct processes eventually decide a block;*
- **Validity**: *the decided block is a block proposed by some process.*

An algorithm has to fulfil these three properties to solve the Byzantine Consensus problem.

Blockchain systems operate over a network, like the Internet, in which the assumption of communication *synchrony*, where every message gets delivered within a known period of time, might be unrealistic. Unfortunately, consensus is known to be unsolvable in asynchronous networks even in the case of a simple crash failure [21]. To cope with this impossibility, various proposals relaxed the guarantees of the classic Byzantine consensus in favor of probabilistic guarantees by exploiting randomization.

3.1 Randomized consensus

Randomization helps bypassing the impossibility result by guaranteeing probabilistic properties instead of deterministic ones. Rabin [39] proposed a solution to the Byzantine consensus in the asynchronous case that terminates in a constant expected time, by combining digital signatures and a trusted leader with randomization. Randomized solutions [3] were also investigated to solve consensus with high probability. Ben-Or's consensus protocol [6] was the first to always ensure agreement but guaranteeing termination with high probability in the crash failure model with up to $f < \frac{n}{2}$ faulty processes where n is the number of participants and f is an upper-bound on the number of failures. Since then, randomisation has been extensively used to help solving consensus [11, 32, 30].

It is not easy to relate the probabilistic guarantees offered by proof-of-work blockchain consensus protocols to the consensus definition used in the distributed computing literature. For example, the Bitcoin analysis relies on the fact that the probability of agreement upon a block at a given index of the blockchain increases exponentially with the depth of the blockchain [33]. In theory, one can thus consider that Bitcoin solves agreement but not termination, as it solves agreement with probability 1 when the execution is infinite. Another way of looking at it is to consider that Bitcoin solves termination but not agreement: if after a finite time or a finite number $i + k$ of blocks $i + k$ have been mined, we say that consensus is reached for block at index i, then the probability that agreement is satisfied is lower than 1: agreement is not guaranteed.

3.2 Termination requirements

There exist, however, important guarantees that are needed by the applications that make use of blockchains. One of these key consensus properties that must be guaranteed for most blockchain applications is termination [25]. For the sake of responsiveness and availability, the application must respond to the client [45]. As an example, a client issuing a transaction to buy goods must eventually receive a response indicating the success or failure of its transaction. As another example, a financial application may require settlement finality, to reduce the risk of insolvency of a participant.[4]

Some blockchain protocols, like Bitcoin-NG [18], have however been proposed with the idea of guaranteeing termination with some probability, hence leaving room for an application to be unresponsive in rare cases. If the termination is not guaranteed deterministically, then the application cannot respond. Other consensus algorithms rely on some leader self-electing itself probabilistically [7]. Some blockchain applications must actually terminate deterministically in order to alleviate the settlement risks observed in financial applications.[5]

[4] http://ec.europa.eu/finance/financial-markets/settlement/index_en.htm.
[5] http://tabbforum.com/opinions/settlement-risks-involving-public-blockchains.

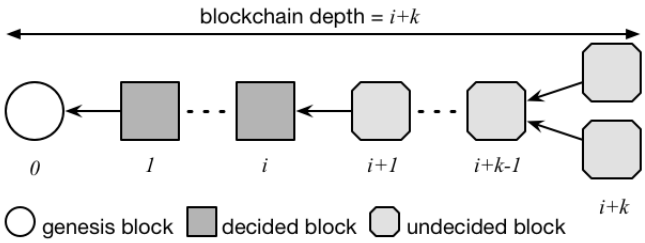


Figure 2: The blockchain structure starts with a genesis block at index 0 and links successive blocks in reverse order of their index; a new block is decided at index $i > 0$ when the blockchain depth reaches $i + k$ (note that a blockchain of depth 0 is the genesis block)

Interestingly, one could propose a different definition from the problem solved by Bitcoin-NG by defining the termination of the Bitcoin consensus protocol [33], which is used in Bitcoin, as follows. One can observe the creation of distinct blocks at the same index of a blockchain as a transient violation of agreement as depicted with the two blocks at index $i + k$ in Figure 2. Under the synchrony assumption, a reorganization (cf. Section 4.3) guarantees however with high probability that the block at index i is uniquely *decided* when the chain depth reaches $i + k$. Garay et al. [22] noted that this probability grows exponentially fast with the depth. Applications can then consider that the depth reaching $i + k$ as the termination of consensus for the block at index i, indicating that the transactions of this block are successfully *committed* [25] and, for example, that goods bought by these transactions can be shipped. Following this reasoning and selecting an appropriate parameter k, one can show that Bitcoin can, in principle (provided that message delays are bounded and that correct processes have a sufficient computational power), solve the Monte Carlo Byzantine consensus problem (described below).

3.3 Monte Carlo consensus

Monte Carlo algorithms are interesting for applications with termination requirements as they guarantee termination even though they may return an incorrect outcome. Monte Carlo consensus was defined as a variant of the Byzantine consensus where safety properties could be violated with non-null probability but where the termination would typically be guaranteed [4]:

Definition 2 (Monte Carlo Byzantine Consensus)**.** *The* Monte Carlo Byzantine Consensus problem *is to guarantee the conjunction of these three properties:*

- **Probabilistic agreement***: no two correct processes decide different blocks with probability at least δ;*
- **Termination***: all correct processes eventually decide a block;*
- **Validity***: the decided block is a block proposed by some process.*

An algorithm has to fulfil these three properties to solve the Monte Carlo Byzantine Consensus problem.

This variant of consensus would guarantee that the blockchain application returns a (sometimes incorrect) result to the client. To avoid being unresponsive, the application could decide of a timeout after which it considers the transaction successful even though the blockchain consensus did not acknowledge this success. For example, a merchant could wait for a predetermined period during which it observes any possible invalidation of the transaction by the blockchain. After this period and if no

invalidation occurred, the transaction is considered valid. Note that there exist variants of the Monte Carlo consensus problem where the validity is also probabilistic [31].

In the worst case scenario, the merchant may be wrong and the transaction may eventually be considered invalid, in which case the merchant will loose goods. Provided that this scenario occurs with a sufficiently small probability over all transactions, the merchant can predetermine its waiting period based on her expected gain over a long series of transactions.

4 Main Blockchain Consensus Algorithms

In this section we build upon Algorithm 1 to explore the differences and similarities of the consensus algorithms of Bitcoin and Ethereum, which are today's predominant blockchain systems.

4.1 Directed Acyclic Graph

Let the *blockchain* be a directed acyclic graph (DAG) $\ell = \langle B, P \rangle$ such that blocks of B point to each other with pointers P and a special block $g \in B$, called the *genesis block*, does not point to any block. There are two important assumptions to guarantee the DAG structure:

1. *Collision-resilience assumption*: The hashing function used for computing the hash of a block is collision resilient and the content of each block is unique.

2. *Unique-pointer-per-block assumption*: Each non-genesis block contains exactly one hash of another block, hence its outdegree is 1.

Algorithm 1 describes the progressive construction of the blockchain at a particular node p_i upon reception of blocks from other processes by simply aggregating the newly received blocks to the known blocks (lines 13–15). As every added block contains a hash to a previous block that eventually leads back to the genesis block indicated by its *parent* field, each block is associated with a fixed index. By convention we consider the genesis block at index 0, and the blocks at k hops away from the genesis block as the blocks at index k. As an example, consider the simple blockchain $\ell_1 = \langle B_1, P_1 \rangle$ depicted in Figure 1(a) where $B_1 = \{g, b_1\}$ and $P_1 = \{\langle b_1, g \rangle\}$. The genesis block g has index 0 and the block b_1 has index 1.

4.2 Forks as disagreements on the blocks at a given index

As depicted by views ℓ_1, ℓ_2 and ℓ_3 in Figures 1(a), 1(b) and 1(c), respectively, processes may have a different views of the current state of the blockchain. In particular, it is possible for two miners p_1 and p_2 to mine almost simultaneously two different blocks, say b_1 and b_2. If neither block b_1 nor b_2 was propagated early enough to processes p_2 and p_1, respectively, then both blocks would point to the same previous block g as depicted in Figures 1(a) and 1(b). Because network delays are not predictable, a third node p_3 may receive the block b_1 and mine a new block without hearing about b_2. The three processes p_1, p_2 and p_3 thus end up having three different local views of the same blockchain, denoted $\ell_1 = \langle B_1, P_1 \rangle$, $\ell_2 = \langle B_2, P_2 \rangle$ and $\ell_3 = \langle B_3, P_3 \rangle$.

We refer to the *global blockchain* as the directed acyclic graph $\ell_0 = \langle B_0, P_0 \rangle$ representing the union of these local blockchain views, denoted by $\ell_1 \cup \ell_2 \cup \ell_3$ for short, as depicted in Figure 1, and more formally defined as follows:

$$\begin{cases} B_0 & = \cup_{\forall i} B_i, \\ P_0 & = \cup_{\forall i} P_i. \end{cases}$$

The point where distinct blocks of the global blockchain DAG have the same predecessor block is called a *fork*. As an example Figure 1(d) depicts a fork with two branches pointing to the same block: g in this example.

In the remainder of this paper, we refer to the DAG as a *tree* rooted in g with upward pointers allowing *children* blocks to point to their *parent* block.

4.3 The main branch selection process

To resolve the forks and define a deterministic state agreed upon by all processes, a blockchain system must select a *main branch*, as a unique sequence of blocks, based on the tree. Building upon the general proof-of-work consensus algorithm (Alg. 1), we present now the characteristics of the Bitcoin consensus algorithm (Alg. 2) [33] and a variant of the Ethereum consensus algorithm (Alg. 3) [46], also called GHOST [42].[6]

4.3.1 The Bitcoin consensus algorithm

The difficulty of the cryptopuzzles used in Bitcoin produces a block every 10 minutes in expectation. The advantage of this long period, is that it is relatively rare for the blockchain to fork because blocks are rarely mined during the time others are propagated to the rest of the processes.

Algorithm 2 The additional field and functions used by the Bitcoin consensus at p_i

```
19: m = 5, the number of blocks to be appended after the block containing
20:     tx, for tx to be committed in Bitcoin

21: get-main-branch()_i:                                   ▷ select the longest branch
22:     b ← genesis-block(B_i)                             ▷ start from the blockchain root
23:     while b.children ≠ ∅ do                            ▷ prune shortest branches
24:         block ← argmax_{c∈b.children}{depth(c)}        ▷ root of deepest subtree
25:         B ← B ∪ {block}                                ▷ update vertices of main branch
26:         P ← P ∪ {⟨block, b⟩}                           ▷ update edges of main branch
27:         b ← block                                      ▷ move to next block
28:     return ⟨B, P⟩                                      ▷ returning the Bitcoin main branch

29: depth(b)_i:                                            ▷ depth of tree rooted in b
30:     if b.children = ∅ then return 1                    ▷ stop at leaves
31:     else return 1 + max_{c∈b.children} depth(c)        ▷ recurse at children
```

Algorithm 2 depicts the Bitcoin-specific pseudocode that includes its consensus protocol to decide on a particular block at some index (lines 21–31) and the choice of parameter m (line 19) explained later in Section 4.4. When a fork occurs, the Bitcoin protocol resolves it by selecting the deepest branch as the main branch (lines 21–28) by iteratively selecting the root of the deepest subtree (line 24). When process p_i is done with this pruning, it obtains the main branch of its blockchain view. Note that the pseudocode for checking whether a block is decided and a transaction committed based on this parameter m is common to Bitcoin and Ethereum, and was presented in lines 13–18 of Alg. 1; only the parameter m used in these lines differ between the Bitcoin consensus algorithm (Alg. 2, line 19) and this variant of the Etherem consensus algorithm (Alg. 3, line 19).

[6] At the time of writing, the Ethereum consensus algorithm in use differs significantly from the GHOST protocol. For the sake of simplicity, we will focus here on the GHOST protocol when referring to the Ethereum consensus algorithm.

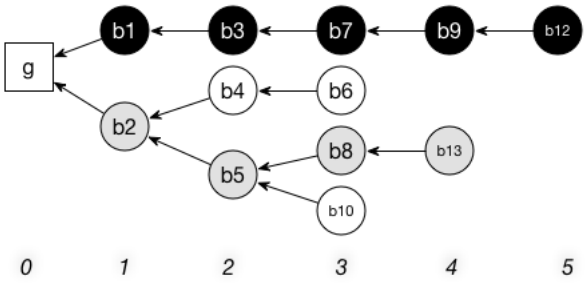


Figure 3: Nakamoto's consensus protocol at the heart of Bitcoin selects the main branch as the deepest branch (in black) whereas the GHOST consensus protocol at the heart of Ethereum follows the heaviest subtree (in grey)

4.3.2 The Ethereum consensus algorithm

As opposed to the Bitcoin protocol, Ethereum generates one block every 12–15 seconds.[7] While it improves the throughput (transactions per second) it also favors transient forks as miners are more likely to propose new blocks without having heard about the latest mined blocks yet. To avoid wasting large mining efforts while resolving forks, Ethereum uses a variant GHOST (Greedy Heaviest Observed Subtree) consensus algorithm that accounts for the, so called *uncles*, blocks of discarded branches. In contrast with the Bitcoin consensus protocol, the GHOST consensus protocol iteratively selects, as the successor block, the root of the subtree that contains the largest number of nodes (cf. Algorithm 3). Note that the current code of Ethereum selects a branch based on the difficulty of the cryptopuzzles solved to obtain the blocks of this branch without comparing the sizes of the subtrees.

Algorithm 3 The additional field and functions used by the Ethereum consensus at p_i

$m = 11$, the number of blocks to be appended after the block containing
tx, for tx to be committed in Ethereum (since Homestead v1.3.5)

get-main-branch$()_i$: ▷ *select the branch with the most nodes*
  $b \leftarrow$ genesis-block(B_i) ▷ *start from the blockchain root*
  while $b.children \neq \emptyset$ **do** ▷ *prune lightest branches*
    $block \leftarrow \text{argmax}_{c \in b.children}\{\text{num-desc}(c)\}$ ▷ *root of heaviest tree*
    $B \leftarrow B \cup \{block\}$ ▷ *update vertices of main branch*
    $P \leftarrow P \cup \{\langle block, b\rangle\}$ ▷ *update edges of main branch*
    $b \leftarrow block$ ▷ *move to next block*
  return $\langle B, P\rangle$. ▷ *returning the Ethereum main branch*

num-desc$(b)_i$: ▷ *number of nodes in tree rooted in b*
  if $b.children = \emptyset$ **then return** 1 ▷ *stop at leaves*
  else return $1 + \sum_{c \in b.children}$ num-desc(c) ▷ *recurse at children*

The main difference between the Bitcoin and Ethereum consensus protocols is depicted in Figure 3, where the black blocks represent the main branch selected by Nakamoto's consensus protocol and the grey blocks represent the main branch selected by GHOST.

[7] This period increases regularly under the influence of a recent algorithm called "the time bomb" that adapts the difficulty of crypto-puzzle.

4.4 Decided blocks and committed transactions

A blockchain system S must define when the block at an index is agreed upon. To this end, it has to define a point in its execution where a prefix of the main branch can be "reasonably" considered as persistent.[8] More precisely, there must exist a parameter m provided by S for an application to consider a block as *decided* and its transactions as *committed*. This parameter is typically $m_{bitcoin} = 5$ in Bitcoin (Alg. 2, line 19) and $m_{ethereum} = 11$ in Ethereum (Alg. 3, line 19). Note that these two choices do not lead to the same probability of success [23] and different numbers are suggested by different applications [35].

Definition 3 (Transaction commit)**.** *Let $\ell_i = \langle B_i, P_i \rangle$ be the blockchain view at node p_i in system S. For a transaction tx to be* locally committed *at p_i, the conjunction of the following properties must hold in p_i's view ℓ_i:*

1. *Transaction tx has to be in a block $b_0 \in B_i$ of the main branch of system S. Formally, $tx \in b_0 \wedge b_0 \in B'_i : c_i = \langle B'_i, P'_i \rangle = \mathsf{get\text{-}main\text{-}branch}()_i$.*
2. *There should be a subsequence of m blocks $b_1, ..., b_m$ appended after block b. Formally, $\exists b_1, ..., b_m \in B_i : \langle b_1, b_0 \rangle, \langle b_2, b_1 \rangle, ..., \langle b_m, b_{m-1} \rangle \in P_i$. (In short, we say that b_0 is* decided.*)*

A transaction tx is committed *if there exists a process p_i where tx is* locally committed.

Property (1) is needed because processes eventually agree on the main branch that defines the current state of accounts in the system—blocks that are not part of the main branch are ignored. Property (2) is necessary to guarantee that the blocks and transactions currently in the main branch will persist and remain in the main branch. Before these additional blocks are created, processes may not have reached consensus regarding the unique blocks b at index j in the chain. This is illustrated by the fork of Figure 1 where processes consider, respectively, the pointer $\langle b_1, g \rangle$ and the pointer $\langle b_2, g \rangle$ in their local blockchain view. By waiting for m blocks were m is given by the blockchain system, the system guarantees with a reasonably high probability that processes will agree on the same block b.

For example, consider a fictive blockchain system with $m_{fictive} = 2$ that selects the heaviest branch (Alg. 3, lines 21–28) as its main branch. If the blockchain state was the one depicted in Figure 3, then blocks b_2 and b_5 would be decided and all their transactions would be committed. This is because they are both part of the main branch and they are followed by at least 2 blocks, b_8 and b_{13}. (Note that we omit the genesis block as it is always considered decided but does not include any transaction.)

5 How Proof-Of-Work Blockchains Can be Unsafe

As a drawback of randomized consensus with deterministic termination, the safety properties of main blockchain systems can be violated. Research efforts were devoted to understand the impact of network delays and mining power distribution on the probability of agreement violations in Bitcoin and Ethereum, leading potentially to double spending, a formalization of which can be find in [1] and [36], respectively. Building upon the tradeoff between termination and agreement mentioned in Section 3.3, a solution would be to make the termination dependent on environmental factors, like network delays and mining power variations in order to guarantee a minimal probability of success.

Unfortunately, existing applications simply rely on fixed parameters regardless of these factors, like waiting for a fixed number m of blocks to be mined in order to reach termination. As we will see in Section 5.1, there are multiple ways Byzantine processes can vary the delays and mining power

[8] In theory, there cannot be consensus on a block at a particular index [21], hence preventing persistence, however, applications have successfully used Ethereum to transfer digital assets based on parameter $m_{ethereum} = 11$ [35].

to double spend in Bitcoin and Ethereum, and some already translated in significant financial losses. As we describe in Sections 5.2 and 5.3, some of these issues are not inherent to the Bitcoin consensus protocol but could also occur with the Ethereum consensus protocol. With the advent of consortium and private blockchains, some of these factors are even simply produced by a misconfiguration of the deployed blockchain systems.

5.1 Attacks against Bitcoin

Traditional attacks against Bitcoin consist of waiting for some external action, like shipping goods, in response to a transaction before discarding the transaction from the main branch. As the transaction is revoked, the issuer of the transaction can reuse the coins of the transaction in another transaction. As the side effects of the external action cannot be revoked, the second transaction appears as a "double spending".

Perhaps the most basic form of such an attack assumes that an application takes an external action as soon as a transaction is included in a block [20, 28, 5]. The first attack of this kind is called Finney's attack and consists of solo-mining a block with a transaction that sends coins to itself without broadcasting it before issuing a transaction that double-spends the same coin to a merchant. When the goods are delivered in exchange of the coins, the attacker broadcasts its block to override the payment to the merchant. The vector76 attack [43] consists of an attacker solo-mining after block b_0 a new block b_1 containing a transaction to a merchant to purchase goods. Once another block b'_1 is mined after b_0, the attacker quickly sends b_1 to the merchant for an external action to be taken. If b'_1 is accepted by the system, the attacker can issue another transaction with the coins spent in the discarded block b_1.

The attacks become harder if the external action is taken after the transaction is committed by the blockchain. Rosenfeld's attack [40] consists of issuing a transaction to a merchant. The attacker then starts solo-mining a longer branch while waiting for m blocks to be appended so that the merchant takes an external action in response to the commit. The attack success probability depends on the number m of blocks the merchant waits before taking an external action and the attacker mining power. However, when the attacker has more mining power than the rest of the system, the attack, also called *majority hashrate attack* or *51-percent attack*, is guaranteed successful, regardless of the value m. To make the attack successful when the attacker owns only a quarter of the mining power, the attacker can incentivize other miners to form a coalition [19] until the coalition owns more than half of the total mining power.

Without a quarter of the mining power, discarding a committed transaction in Bitcoin requires additional power, like the control over the network. It is well known that delaying network messages can impact Bitcoin [17, 38, 42, 24, 37]. Decker and Wattenhoffer already observed that Bitcoin suffered from block propagation delays [17]. Godel et al. [24] analyzed the effect of propagation delays on Bitcoin using a Markov process. Garay et al. [22] investigated Bitcoin in the synchronous communication setting. Pass et al. extended the analysis for when the bound on message delivery is unknown and showed in their model that the difficulty of Bitcoin's crypto-difficulty has to be adapted depending on the bound on the communication delays [38]. This series of work reveal an important limitation of Bitcoin: delaying propagation of blocks can waste the computational effort of correct processes by letting them mine blocks unnecessarily at the same index of the chain. In this case, the attacker does not need more mining power than the correct miners, but simply needs to expand its local blockchain faster than the growth of the longest branch of the correct blockchain.

Ethereum proposed the GHOST protocol to cope with this issue [42]. The idea is simply to account for the blocks proposed by correct miners in the multiple branches of the correct blockchain to select the main branch. As a result, growing a branch the fastest is not sufficient for an attacker of Ethereum to be able to double spend.

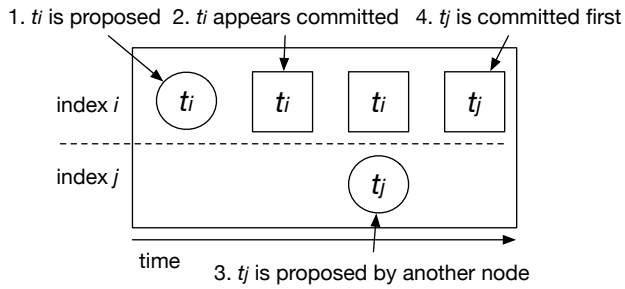


Figure 4: The blockchain anomaly: a first client issues t_i that gets successfully mined and committed then a second client issues t_j, with t_j being conditional to the commit of t_i (note that $j \geq i + k$ for t_i to be committed before t_j gets issued), but the transaction t_j gets finally reorganized and successfully committed before t_i, hence violating the dependency between t_i and t_j

5.2 The blockchain anomaly in Ethereum

We identified the Blockchain anomaly [35] that prevents someone from executing dependent transactions like "Bob transfers some coins to Carole only if it received coins from Alice" and that allows an attacker to double spend. This issue is named the Blockchain anomaly after the Paxos anomaly [8] because it stems from a reordering of decided blocks. In contrast to the Paxos anomaly, the Blockchain anomaly occurs even if Bob waits for the reception of the money from Alice to be successfully committed before transferring to Carole.

The Blockchain anomaly was experimented on an Ethereum private chain with 2 mining pools running geth v1.4.0 in a controlled network. Although two miners mine on the same chain starting from the same genesis block, a long enough delay in the delivery of messages could lead to having the miners seemingly agree separately on different branches containing more than m blocks each, for any $m \in \mathbb{N}$. When messages get finally delivered, the results of the disagreement creates inconsistencies, like the reordering or deletion of transactions from previously decided blocks.

Precisely because the length of the branch could be adjusted to any m, it guarantees that there is no way for an application to choose m' sufficiently large to guarantee that consensus is reached. To take the classic example of exchanges, choosing $m'_{btc} = 5$ for Bitcoin and $m'_{eth} = 11$ for Ethereum cannot be sufficient, as there exist a $m = 12$ for which the Blockchain anomaly can occur. This anomaly is dramatic as it can lead to simple double-spending attacks within a network where users have an incentive to maximize their profits—in terms of coins or arbitrary ownership.

Figure 4 depicts the blockchain anomaly, where a transaction t_i is proposed as part of a block at index i from the standpoint of some processes. Based on this observation, one waits for t_i to commit before proposing a new transaction t_j. Again, one can imagine a simple scenario where "Bob transfers an amount of money to Carole" (t_j) only if "Bob had successfully received some money from Alice" (t_i) before. However, once these processes get notified of another branch of committed transactions, they decide to reorganize the branch to resolve the fork. The reorganization removes the committed transaction t_i from slot i. Later, the transaction t_j is successfully committed in slot i. A reproducible distributed execution is described in detail in [35].

This anomaly translates into a scenario that is counter-intuitive for the user of Ethereum: a transaction may not persist even though it is committed. This illustrates that in addition to the attacks against Bitcoin, there exist attacks against other form of proof-of-work consensus algorithms, like Ethereum's. Moreover, this scenario is realistic in the context of private chains where participating competitors have direct access to some of the resources. The Blockchain anomaly stems from the fact that in a private

chain the reward system does not necessarily incentivize many processes to mine correctly. Note that in the R3 experiments not all processes were mining because it was decided they would not do so [34].

5.3 The balance attack against Ethereum

In the *Balance Attack* [34], an attacker transiently disrupts communications between subgroups of Ethereum miners of similar mining power. During this time, the attacker issues transactions in one subgroup, say the *transaction subgroup*, and mines blocks in another subgroup, say the *block subgroup*, up to the point where the tree of the block subgroup outweighs, with high probability, the tree of the transaction subgroup.

The balance attack is simple: after the attacker introduces a delay between correct subgroups of equivalent mining power, it simply issues transactions in one subgroup. The attacker then mines sufficiently many blocks in another subgroup to ensure with high probability that the subtree of another subgroup outweighs the transaction subgroup's. Even though the transactions are committed, the attacker can rewrite with high probability the blocks that contain these transactions by outweighing the subtree containing this transaction.

Note that one could benefit from delaying messages only between the merchant and the rest of the network by applying the eclipse attack [27] to Ethereum. Eclipsing one node of Bitcoin appeared, however, sufficiently difficult: it requires to restart the node's protocol in order to control all the logical neighbors the node will eventually try to connect to. While a Bitcoin node typically connects to 8 logical neighbors, an Ethereum node typically connects to 25 nodes, making the problem even harder. Another option would be to isolate a subgroup of smaller mining power than another subgroup, however, it would make the attack only possible if the recipients of the transactions are located in the subgroup of smaller mining power. Although possible this would limit the generality of the attack, because the attacker would be constrained on the transactions it can override.

Note that the Balance Attack inherently violates the persistence of the main branch prefix and is enough for the attacker to double spend. The attacker has simply to identify the subgroup that contains merchants and create transactions to buy goods from these merchants. After that, it can issue the transactions to this subgroup while propagating its mined blocks to at least one of the other subgroups. Once the merchant shipped goods, the attacker stops delaying messages. Based on the high probability that the tree seen by the merchant is outweighed by another subtree, the attacker could reissue another transaction transferring the exact same coin again.

6 Defining the Blockchain Byzantine Consensus

Perhaps the main reasons why large-scale blockchain systems suffer from such inconsistencies is that the existing consistent consensus solutions are inefficient due to the restrictive problem that they solve. In particular, safe blockchain typically use off-the-shelf algorithms (e.g., PBFT, BFTSmart) that solves the classic Byzantine consensus (Definition 1) as a blackbox. This typically prevents them from scaling to tens of nodes.

In the light of this limitation, we revisited the Byzantine consensus problem as a problem tailored to blockchain system, called the *Blockchain Consensus* [14]. Its distinction relies on its validity property that makes use of the validation inherent to blockchain system: a blockchain-specific `valid` predicate that guarantees that a set of transaction can be executed. Assuming that each correct process proposes a valid value, each of them has to decide on a value in such a way that the following properties are satisfied.

Definition 4 (Blockchain Byzantine Consensus)**.** *The* Blockchain Byzantine Consensus *problem is to guarantee the conjunction of these three properties for a given index:*

- **Agreement**: *no two correct processes decide different blocks;*
- **Termination**: *all correct processes eventually decide a block;*
- **Validity**: *a decided block is valid, it satisfies the predefined predicate* `valid`.

An algorithm has to fulfil these three properties to solve the Blockchain Byzantine Consensus problem.

As far as we know, the only algorithm that solves the Blockchain Byzantine Consensus problem is called the Democratic BFT (DBFT) and was first formalized in [15]. The reason why this definition of consensus is better suited to blockchain is twofold. First, the valid predicate allows the blockchain to decide a block of transactions that was proposed by Byzantine participants. This difference is possible thanks to the use of the `valid` predicate that defines the validity of a block proposed by a Byzantine participant. Without this `valid` predicate, the decided value could not be one of the values proposed by a Byzantine as these are undefined. Second, the decided value does not need to be one of the proposed value. This allows to decide a number of transactions that grows potentially with the number of participants. To solve the classic Byzantine consensus, only one of the proposed block could be decided, hence limiting the number of decided block to 1 out of $n - t$ blocks of transactions proposed by correct participants. To solve the Blockchain consensus, however, the decided block could represent the union of all the $n - t$ blocks proposed by correct participants.

7 Refining the Blockchain Model for Consortiums

As we discussed previously, the risk of safety violation of main blockchain systems stems from the impossibility of solving consensus deterministically in the general case, which also applies to the more general Blockchain Byzantine consensus (Definition 4). There are however solutions that consist of restricting the model by listing additional assumptions under which an alternative blockchain system could be made both safe and live. The *consortium* model is getting traction for allowing a pre-selected set, called consortium, of participants typically part of different institutions control the consensus protocol of the blockchain system.

7.1 Differences between consortium and classic blockchain models

The main differences with the public blockchain model are listed below.

1. **Permissioned**: only a specified set of institutions can participate in the consensus of the consortium blockchain. The fact that each participant needs a permission to participate in the consensus does not prevent other users to potentially access the current state of the blockchain, they simply cannot take part of the decision process. The appealing aspects of this consortium is that the decision is not controlled by a leader [26] or a single institution as in the case of fully-private blockchains. This is also in contrast with the permissionless Bitcoin and Ethereum main chain in which any participant connected to Internet can join at any time, and alleviates the problem of having an uncontrollable amount of nodes wasting resources.

2. **Global knowledge**: given that the membership is pre-determined, one can reasonably assume that most participants are aware of the exact list of the n participants of the consortium. Consequently, any participant that lags behind, simply needs to contact a majority or a quorum of participants to catch up with the most up-to-date system size n. Moreover, this fixed list of participants naturally prevents an attacker from executing a Sybil attack by forging multiple identities it can control.

3. **Bound on the number of failures**: given that the list of participants is known, one can reasonably assume that a malicious participant cannot convince the consortium to introduce a large number of fake identities in comparison to the consortium size. Moreover, one can assume that new participants go through a detailed KYC (know-your-customer) process before getting the permission to join the consortium. This makes it realistic to limit coalitions of f malicious participants to $f << n$ at the same time.

Despite these differences, the consortium blockchain model is close to the original blockchain model as we explain below.

7.2 Similarities between consortium and classic blockchain models

The consortium model is still very close to the general blockchain model.

1. The **failure model** is the same as in the classic model. It is still necessary to tolerate Byzantine failures in a consortium model as the participating institutions can have conflicting interests, and the blockchain should protect from the possible misbehavior of a participant.

2. The **communication model** is also the same as in the classic model. These institutions may be located in different regions of the globe and typically communicate through internet when issuing transactions. The Internet is unpredictable and the delay of a message cannot be known in advance. In particular, the Internet network is shared by machines external to the consortium and is subject to large localized failures due to disasters, it is thus impossible to control or even anticipate traffic disruptions, congestions and delays.

Provided that $f < \frac{n}{3}$, practical Byzantine fault tolerant solutions could be used realistically to solve the Byzantine consensus in the consortium blockchain model and without the need for proof-of-work. Of course, practical Byzantine fault tolerant solutions remain quite limited for several reasons: (i) they usually require a leader election that is difficult to implement and that conflicts with the inherent decentralization aim of blockchains: in particular it is impossible to elect a correct leader as a Byzantine leader could act correctly up to the point where it gets elected, (ii) they often employ complex techniques to circumvent the impossibility result like a global random coin that returns the same random value to any process and whose values cannot be anticipated by Byzantine processes, and (iii) they typically rely on costly public-key cryptosystems to guarantee authentication.

7.3 Consortium blockchain systems

Consortium blockchains are getting popular for companies to benefit from the blockchain properties in a controlled environment where the consensus participants are fixed and well-known. While not full-fledged yet, some blockchain systems were proposed for companies to run a consortium blockchain.

7.3.1 Ripple

The consensus protocol of Ripple, the third largest digital currency in market capitalization, was proposed as a white paper and uses mutually interesting sets of replicas, also known as quorums [41]. The protocol bootstraps with a hard-coded list of initial replicas. Each node requests a different list of replicas, also called a *unique node list (UNL)*, and waits until a quorum, which represents at least 80% of this list, answers. It also requires minimal connectivity and an intersection size among UNLs that represents at least 20% of each UNL. However, there has been a debate wether the assumptions of the Ripple consensus were suffient to implement consensus. In particular, the intersection property alone was shown insufficient to solve consensus and that stricly more than 40% was actually required [2].

7.3.2 The Hyperledger fabric

IBM is a key partner in the Hyperledger project [12], a recent industry-wide collaborative effort to develop an open-source blockchain. Although the current version of the Hyperledger codebase (v0.6) features a naive consensus approach relying on a central server for testing purpose, the next generation of Hyperledger, is expected to feature the practical fault tolerant Byzantine protocol [13] and a variant of Apache Kafka[9]. PBFT [13] and Kafka are being implemented as modular consensus protocol one can plug to Hyperledger. Hyperledger also features a subledger abstraction that allows partners to collaborate within a consortium blockchain without revealing the content of the blockchain to other users.

7.3.3 The R3 Consortium

We experimented a distributed system running Ethereum in similar settings as R3, a consortium of more than 70 world-wide financial institutions. In January 2016, R3 consisted of eleven banks and successfully collaborated in deploying an Ethereum private chain to perform transactions.[10] Since then, R3 has grown and kept experimenting Ethereum[11] and other technologies while the concept of consortium private chain gained traction for its ability to offer a blockchain system among multiple companies in a private and controlled environment. R3 has just released their own Corda framework. As opposed to a fully private chain scenario, the consortium private chain involves different institutions possibly competing among each other. As they can be located at different places around the world, they typically use Internet to communicate. The R3 consortium has been experimenting Ethereum since more than half a year now and our discussion with the R3 consortium indicated that they did not investigate the dependability of the GHOST consensus protocol. The Balance attack was demonstrated in the R3 testbed setting [34] but R3 also worked with various blockchain approaches including Ripple, Axoni and Symbiont. At the time of writing, R3 has just released Corda [10] as a proposed solution for private chains. Corda does not yet recommend a particular consensus protocol but mentions Byzantine fault tolerance consensus algorithms and favors modularity by allowing to plug any consensus protocol instead [10].

7.4 The Red Belly Blockchain

Recently, a new blockchain appeared particularly promising, the Red Belly Blockchain.[12] It relies on the Democratic BFT [15] that solves the Blockchain Byzantine Consensus problem (Definition 4). Because it does not rely on an off-the-shelf classic Byzantine consensus algorithm, the Red Belly Blockchain already scales to more than 100 consensus participants and handle a workload of more than 400 thousand transactions per second, hence tolerating a potentially much larger number of blockchain participants issuing transactions and requesting balances than existing blockchains. In contrast with other large-scale blockchains, the Red Belly Blockchain achieves fast settlement (typically within 3 seconds) because it does not need any proof-of-work. Finally, an interesting aspect of the Red Belly Blockchain is its difference with consortium blockchains listed above, whose consortium is typically static. The Red Belly Blockchain offers a dynamic consortium that works through cooptation: the current consortium members decide upon proposed consortium changes and, once decided, the new consortium takes over the responsibility of participating in further consensus instances to decide upon blocks or new consortia. With this model, any participant of the blockchain can eventually

[9] https://kafka.apache.org.

[10] http://www.ibtimes.co.uk/r3-connects-11-banks-distributed-ledger-using-ethereum-microsoft-azure-1539044.

[11] http://www.coindesk.com/r3-ethereum-report-banks/.

[12] http://poseidon.it.usyd.edu.au/~concurrentsystems/rbbc/.

become a consensus participants in charge of deciding.

8 Conclusion

While the blockchain technology is reshaping ownership tracking through distributed ledgers, it remains difficult for blockchain users to understand the guarantees this technology has to offer. This paper describes the causes of this difficulty in mainstream proof-of-work blockchain systems, namely Bitcoin and Ethereum. One cause is the probabilistic nature of its consensus algorithms: although it appears that one should wait longer to increase the probability of agreement in case of network delays, most applications rely on a fixed predicate to define the termination of consensus. Another cause is that users have started deploying blockchain protocols in either a private or a consortium context, often involving fewer miners with a different distribution of the mining power and where network delays can be artificially introduced.

While the recent redefinition of the consensus problem in the context of blockchain helps addressing the major tradeoff between consistency and performance, it is crucial to design new provable algorithms especially tailored for blockchains and validate them through large-scale experimentations.

References

[1] Emmanuelle Anceaume, Thibaut Lajoie-Mazenc, Romaric Ludinard, and Bruno Sericola. Safety analysis of bitcoin improvement proposals. In *15th IEEE International Symposium on Network Computing and Applications, NCA 2016*, pages 318–325, 2016.

[2] Frederik Armknecht, Ghassan O. Karame, Avikarsha Mandal, Franck Youssef, and Erik Zenner. Ripple: Overview and outlook. In Mauro Conti, Matthias Schunter, and Ioannis Askoxylakis, editors, *Trust and Trustworthy Computing*, volume 9229 of *Lecture Notes in Computer Science*, pages 163–180. Springer International Publishing, 2015.

[3] James Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2-3):165–175, 2003.

[4] James Aspnes. Faster randomized consensus with an oblivious adversary. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 1–8, 2012. Personal communication.

[5] Tobias Bamert, Christian Decker, Lennart Elsen, Roger Wattenhofer, and Samuel Welten. Have a snack, pay with bitcoins. In *13th IEEE International Conference on Peer-to-Peer Computing, IEEE P2P 2013, Trento, Italy, September 9-11, 2013, Proceedings*, pages 1–5, 2013.

[6] Michael Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, PODC '83, pages 27–30, New York, NY, USA, 1983. ACM.

[7] Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. Technical Report 919, IACR Cryptology ePrint Archive, 2016.

[8] Ken Birman, Dahlia Malkhi, and Robbert Van Renesse. *Guide to Reliable Distributed Systems: Building High-Assurance Applications and Cloud-Hosted Services*, chapter Appendix A: Virtually Synchronous Methodology for Building Dynamic Reliable Services, pages 635–671. Springer London, London, 2012.

[9] Adam Black. Hashcash - a denial of service counter-measure. Technical report, Cypherspace, 2002. http://www.hashcash.org/papers/hashcash.pdf.

[10] Richard Gendal Brown, James Carlyle, Ian Grigg, and Mike Hearn. Corda: An introduction, 2016.

[11] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In *Proc. 21st Annual International Cryptology Conference (CRYPTO)*, pages 524–541, 2001.

[12] Christian Cachin. Architecture of the hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers (DCCL'16)*, 2016.

[13] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[14] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. Blockchain consensus. In *19ème rencontres francophones sur les aspects algorithmiques de télécommunications (AlgoTel'17)*, 2017.

[15] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. (leader/randomization/signature)-free byzantine consensus for consortium blockchains. Technical Report 702.03068v2, arXiv, Feb 2017.

[16] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gun Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized blockchains, February 2016.

[17] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *Proc. of the IEEE International Conference on Peer-to-Peer Computing*, pages 1–10, 2013.

[18] Ittay Eyal, Adem Efe Gencer, Emin Gn Sirer, and Robbert van Renesse. Bitcoin-NG: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 45–59, 2016.

[19] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, pages 436–454, 2014.

[20] Hal Finney. Finney's attack, February 2011.

[21] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.

[22] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Proceedings of the 34th Annual International Conference on the Theory and Applications of Cryptographic Technique (EUROCRYPT)*, pages 281–310, 2015.

[23] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 3–16, 2016.

[24] J. Göbel, H.P. Keeler, A.E. Krzesinski, and P.G. Taylor. Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay. *Performance Evaluation*, Juy 2016.

[25] Vincent Gramoli. On the danger of private blockchains. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers (DCCL'16)*, 2016.

[26] Vincent Gramoli, Len Bass, Alan Fekete, and Daniel Sun. Rollup: Non-disruptive rolling upgrade with fast consensus-based dynamic reconfigurations. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 27(9):2711–2724, Sep 2016.

[27] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *24th USENIX Security Symposium*, pages 129–144, 2015.

[28] Ghassan Karame, Elli Androulaki, and Srdjan Capkun. Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin. *IACR Cryptology ePrint Archive*, 2012:248, 2012.

[29] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[30] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of BFT protocols. In *Proc. of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[31] Andrew Miller and Joseph J. LaViola Jr. Anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin. Technical Report CS-TR-14-01, University of Central Florida, April 2014.

[32] A. Mostéfaoui, H. Moumen, and M. Raynal. Signature-free asynchronous binary byzantine consensus with $t < n/3$, $o(n^2)$ messages, and $o(1)$ expected time. *Journal of ACM*, 62(4).

[33] Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system, 2008. http://www.bitcoin.org.

[34] Christopher Natoli and Vincent Gramoli. The balance attack against proof-of-work blockchains: The R3 testbed as an example. Technical Report 1765133, arXiv, 2016.

[35] Christopher Natoli and Vincent Gramoli. The blockchain anomaly. In *Proceedings of the 15th IEEE International Symposium on Network Computing and Applications (NCA'16)*, pages 310–317, Oct 2016.

[36] Christopher Natoli and Vincent Gramoli. The balance attack or why forkable blockchains are ill-suited for consortium. In *Proceedings of the 47th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'17)*, June 2017.

[37] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 305–320, 2016.

[38] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. Technical Report 454, Crytology ePrint Archive, 2016.

[39] Michael O. Rabin. Randomized byzantine generals. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, SFCS '83, pages 403–409, 1983.

[40] Meni Rosenfeld. Analysis of hashrate-based double-spending, 2012.

[41] David Schwartz, Noah Youngs, and Arthur Britto. The Ripple consensus protocol, 2014. https://ripple.com/files/ripple_consensus_whitepaper.pdf.

[42] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, pages 507–527, 2015.

[43] vector76. The vector76 attack, August 2011.

[44] Marco Vukolíc. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *Proceedings of the IFIP WG 11.4 Workshop on Open Research Problems in Network Security (iNetSec 2015)*, LNCS, pages 112–125, 2016.

[45] Ingo Weber, Vincent Gramoli, Mark Staples, Alex Ponomarev, Ralph Holz, An Binh Tran, and Paul Rimba. On availability for blockchain-based systems. In *Proceedings of the 36th International Symposium on Reliable Distributed Systems (SRDS'17)*. IEEE, Sep 2017.

[46] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2015. Yellow paper.

[47] Xiwei Xu, Cesare Pautasso, Liming Zhu, Vincent Gramoli, Alexander Ponomarev, An Binh Tran, and Shiping Chen. The blockchain as a software connector. In *13th Working IEEE/IFIP Conference on Software Architecture, WICSA 2016, Venice, Italy, April 5-8, 2016*, pages 182–191, 2016.

View publication stats