Chapter 2

Blockchain Fundamentals

I would be surprised if 10 years from now we're not using electronic currency in some way, now that we know a way to do it that won't inevitably get dumbed down when the trusted third party gets cold feet.

Satoshi Nakamoto

2.1 Introduction

In this chapter, we study mainstream blockchain systems like Bitcoin [Nak08]. We look at how their proof-of-work mechanism can limit the power of the malicious nodes and how their consensus protocol copes with forks by pruning branches in order to converge to a unique chain of blocks. As we presented in Chapter 1, it is possible to exploit these forks to steal assets by double spending. It is thus important when building blockchains to think about all the actions that malicious participants could take to steal assets. This notion of malicious participant that can act arbitrarily has been called "Byzantine" in the distributed computing literature.

2.2 Failures and communication

The system comprises n processes that are *asynchronous*, that is each process computes at its own speed. We consider the message passing model, where processes communicate by sending and receiving messages. For simplicity we consider that processes communicate through point-to-point reliable channel. This means that any pair of processes is connected by a bidirectional channel so that upon delivery of a message, a process can identify the process that sent the message. Also, these channels are reliable in that they do not lose, create, duplicate or modify the messages. We will explain how to relax this assumption in Chapter 5. When not specified, we consider that the communication is asynchronous, that is there is no bound on the time it takes for a message to reach its destination but this time is finite.

In general a system *fails* if it cannot meet its promises and a distributed system of machines is typically a system that can fail if one of its machines fail.[1] Here we consider two types of failure models, the crash failure model in which up to f processes can experience a crash failure or the Byzantine failure model in which up to f processes can experience a Byzantine failure. Each algorithm we will present will either work in one of these two failure models or where no failures occur. More precisely, a process can experience only two types of failures:

- Crash failure: A node experiences a crash failure or fail by crashing if it halts but behaved correctly until it halts. Once a node has crashed, then it cannot do anything, it cannot even recover and send messages again with the same identity as before.
- Byzantine failure: A Byzantine node is a node that behaves arbitrarily: it can crash, fail to send or receive messages, send arbitrary messages, start in an arbitrary state, perform arbitrary state transitions, etc.

We refer to a node that experiences a failure as a *faulty* node, and any non-faulty node is a *correct* node. It is interesting to note that because Byzantine

[1] http://lamport.azurewebsites.net/pubs/distributed-system.txt.

nodes can act arbitrarily, they can collude to make the system fail. Typically, by behaving arbitrarily Byzantine nodes can make more harm to the system than the crash nodes and they can even simulate crash nodes. So any algorithm that works in a Byzantine failure model, also works in the crash failure model.

Also, as each pair of nodes is connected by a channel, no Byzantine node can impersonate another node. Byzantine nodes can control the network by modifying the order in which messages are received, but they cannot postpone forever message receptions.

2.3 Properties of consensus

As mentioned before, the consensus problem has been an important research problem of distributed computing for the past fourty years. It finds applications in maintaining the consistency of a replicated state machine and helps totally ordering messages in a distributed system of machines.

Blockchain systems aim at solving the consensus problem, so that for a given index all correct nodes agree on a unique block of transactions at this index. As mentioned in Section 1 nodes may propose different blocks at the same index; this is generally observed with a fork. The classic definition of consensus is defined along three properties.

Definition 1 (Consensus) *Assuming that each correct node proposes a value, each of them has to decide on a value in such a way that the following properties are satisfied.*

1. *Validity: any decided value is a proposed value.*
2. *Agreement: no two correct nodes decide differently.*
3. *Termination: every correct node eventually decides.*

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

2.4 Impossibility to solve consensus in asynchronous networks

It is important to note that consensus cannot be solved if the communication is asynchronous and there are failures [FLP85]. More precisely, there is no consensus algorithm ensuring both safety and liveness properties in fully asynchronous message-passing systems in which even a single process may crash. As the crash failure model is less severe than the Byzantine failure model, the consensus impossibility remains true if processes may commit Byzantine failures. Although one might be tempted to try to design a protocol that does not always terminate but always guarantees the safety properties (validity and agreement) of consensus, this has recently been shown to be impossible as neither liveness nor safety can be ensured [CGG19]. To cope with this impossibility, various proposals relaxed the guarantees of the classic Byzantine consensus in favor of probabilistic guarantees by exploiting randomization, failure detectors or additional synchrony assumptions.

2.4.1 Failure detectors

To solve consensus, researchers have relied on failure detectors. In the crash-failure model, the weakest class of failure detectors that allows to solve the consensus is $\diamond S$ [CT96] that is equivalent to the eventual leader failure detector [CHT96]. Its eventual accuracy property guarantees that there is a time after which there is a non-faulty process that is never suspected by the non-faulty processes. Building upon this guarantee, several consensus algorithms were proposed to solve consensus, namely, the processes proceed in asynchronous rounds managed by a pre-determined leader or coordinator that tries to impose a value as the decision. Unfortunately, $\diamond S$ cannot be easily implemented in the Byzantine failure model.

2.4.2 Randomized consensus

Randomization can help solve consensus in the Byzantine failure model. Instead of terminating deterministically, randomized consensus algorithms terminate with a probability that tends to 1 as correct processes take steps. Most randomized Byzantine consensus algorithms are binary in that the set of values that can be proposed is $\{0, 1\}$. Randomized Byzantine consensus algorithms typically use "local" coins—one coin per node—or a "common" coin—a shared coin that returns the same value at all nodes. This randomization is generally used to prevent Byzantine processes from anticipating the value returned by the coin to force some correct nodes to diverge by adopting the opposite of this value. Randomized Byzantine consensus can either terminate with probability $(1 - \varepsilon)$ for some non-null but negligible ε or can be almost-surely terminating in that the probability of not terminating is asymptotically null. Probabilistic consensus makes some probabilistic assumptions, assuming for example that the scheduler is fair. This allows to solve consensus probabilistically without random coins as long as the number of Byzantine processes among n processes is $t < n/3$.

2.4.3 Deterministic termination

Probabilistic termination is sometimes not enough. In fact, blockchain applications often need to be available, and thus need to be guaranteed to commit transactions. As an example, a financial application may require settlement

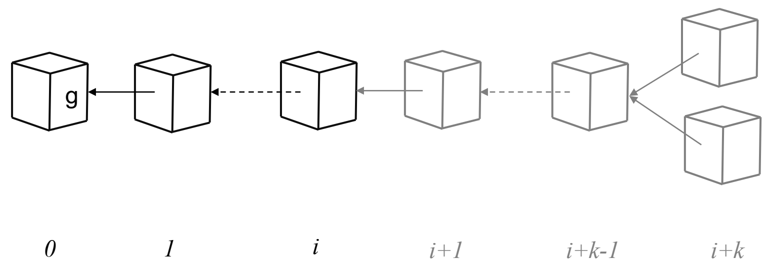

Figure 2.1: A new block is decided at index $i > 0$ when the blockchain depth reaches $i + k$ (note that a blockchain of depth 0 is the genesis block)

finality, to reduce the risk of insolvency of a participant.[2] Blockchains, like Bitcoin-NG [EGSvR16], guarantee termination with some probability, hence leaving room for an application to be unresponsive in rare cases. Other consensus algorithms rely on some leader self-electing itself probabilistically [BPS16]. 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 2.1. Under the synchrony assumption, one can guarantee that in Bitcoin the block at index i is uniquely *decided* with high probability when the chain depth reaches $i + k$. Note that researchers already noted that this probability grows exponentially fast with k [GKL15]. And the recommendation of the Bitcoin project members is to use $k = 5$ or 6 confirmations in case the transfers are of "high" value.[3] (The value $k = 11$ or 12 confirmations was suggested for Ethereum by one of its inventors.[4])

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. In the worst case scenario, the merchant may be wrong and the transaction may eventually be considered invalid, in which case the merchant will lose 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.

[2]http://ec.europa.eu/finance/financial-markets/settlement/index_en.htm.

[3]https://bitcoin.org/en/you-need-to-know.

[4]https://ethereum.stackexchange.com/questions/183/how-should-i-handle-blockchain-forks-in-my-dapp/203.

2.4.4 Additional synchrony

The classic approach to solve consensus deterministically is thus to assume additional synchrony. Another way to solve consensus is to assume *synchronous communication* or synchrony, where every message gets delivered within a known period of time. This means that the algorithm can used this upper bound on the time it takes to deliver a message in order to work. In particular, one could potentially use this bound to detect a failure if it does not get a response from a message it has sent two message delays ago. Another impossibility result indicates that consensus cannot be solved if $n/3$ or more processes are Byzantine. This result applies even if synchrony is assumed as long as there is no authentication but more generally applies regardless of authentication in the asynchronous model. Blockchain systems operate over a network, like the Internet, in which the assumption of communication synchrony is unrealistic. A more realistic assumption is *partially synchronous communication* also referred to as partial or eventual synchrony. Communication is partially synchronous if after some finite Global Stabilization Time (GST), there is an upper bound Δ on message transfer delays. Note that, as opposed to the synchrony assumption, the bound cannot be used by an algorithm as it is unknown.

2.4.5 Impossibility to solve consensus with too many failures

Another impossibility result states that consensus cannot be solved among n nodes if the number of Byzantine failures f is greater or equal to $n/3$ [LSP82]. Note that this is true even if we assume partial synchrony. It is also true if we assume synchrony as long as we do not have authentication—meaning that the processes cannot use signatures. As we have seen in Chapter 1, signatures are already implemented using public-key cryptosystem in blockchain systems. Consequently one could use the same technique within a protocol to solve consensus even in the presence of less than $n/2$ Byzantine processes. However, it is important to note that these results only apply to the Byzantine failure model. Note that in the crash failure model, where we consider that f is an upper-bound on the number of crash failures and that there are no Byzantine failures, consensus is easier to solve. In particular, consensus in the crash failure model can be solved among n nodes if the number of crash failures f is strictly lower than $n/2$.

2.5 Proof of work and mining

Miners have the role of creating blocks, by sometimes provably solving a hashcash crypto-puzzle [Bla02]. Given a global threshold and the block of largest index the miner knows, trying to solve a crypto-puzzle consists of repeatedly selecting a nonce and applying a pseudo-random function to this block and the selected nonce until a result lower than the threshold is obtained. Upon success the miner creates a block that contains the successful nonce 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. As there is no known strategy to solve the crypto-puzzle, the miners simply keep testing whether randomly chosen numbers solve the crypto-puzzle. The mining power is thus expressed in the number of hashes the miner can test per second, or H/s for short. The *difficulty* of this crypto-puzzle, defined by the threshold, limits the rate at which new blocks can be generated by the network.

2.5.1 Proposing to the consensus

Miners initiate the consensus through a propose function depicted at lines 12–18 of Alg. 2 allowing them to propose new blocks. Processes decide upon a new block at a given index at line 24 depending on a function get-main-branch that is specific to the type of proof-of-work blockchain system in use (cf. Algorithms 3 and 4 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 2.2.

Algorithm 2 The general proof-of-work blockchain consensus algorithm at process p_i

```
 6: $\ell_i = \langle B_i, P_i \rangle$, the local blockchain at node $p_i$ is a directed acyclic
 7:    graph of blocks $B_i$ and pointers $P_i$
 8: $b$, a block record with fields:
 9:    *parent*, the block preceding $b$ in the chain, initially $\bot$
10:    *pow*, the proof-of-work nounce of $b$ that solves the cryptopuzzle, initially $\bot$
11:    *children*, the successor blocks of $b$ in the chain

12: propose$()_i$:
13:    **while** true **do**
14:       $nounce$ = local-random-coin$()$
15:       create block $b$ : $b.parent$ = last-block$(\ell_i)$ and $b.pow = nounce$
16:       **if** solve-cryptopuzzle$(nounce, b)$ **then**
17:          broadcast$(\langle \{b\}, \{\langle b, b.parent \rangle\} \rangle)$
18:          break$()$

19: deliver$(\langle B_j, P_j \rangle)_i$:
20:    $B_i \leftarrow B_i \cup B_j$
21:    $P_i \leftarrow P_i \cup P_j$
22:    $\langle B_i', P_i' \rangle \leftarrow$ get-main-branch$()$
23:    **if** $(b_0 \in B_i') \wedge (\exists b_1, ..., b_m \in B_i : \langle b_1, b_0 \rangle, \langle b_2, b_1 \rangle ..., \langle b_m, b_{m-1} \rangle \in P_i)$ **then**
24:       decide$(b_0)$
```

2.5.2 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.[5] 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. 3, line 25) and $m_{ethereum} = 11$ in Ethereum (Alg. 4, line 25). Note that these two choices do not lead to the same probability of success [GKW^{+}16] and different numbers are suggested by different applications [NG16b].

Definition 2 (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 correct 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 1.2 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. 4, lines 27–34) as its main branch. If the blockchain state was the one depicted in Figure 2.2, 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] In theory, there cannot be consensus on a block at a particular index [FLP85], hence preventing persistence, however, applications have successfully used Ethereum to transfer digital assets based on parameter $m_{ethereum} = 11$ [NG16b].

2.6 Resolving forks

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. 2), we present now the characteristics of the Bitcoin consensus algorithm (Alg. 3) [Nak08] and a variant of the Ethereum consensus algorithm (Alg. 4) [Woo15], also called GHOST [SZ15].

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 3 The additional field and functions used by the Nakamoto consensus algorithm at p_i

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

27: get-main-branch$()_i$:
28:     $b \leftarrow$ genesis-block$(B_i)$
29:     while $b.children \neq \emptyset$ do
30:         $block \leftarrow \text{argmax}_{c \in b.children}\{\text{depth}(c)\}$
31:         $B \leftarrow B \cup \{block\}$
32:         $P \leftarrow P \cup \{\langle block, b \rangle\}$
33:         $b \leftarrow block$
34:     return $\langle B, P \rangle$

35: depth$(b)_i$:
36:     if $b.children = \emptyset$ then return $1$
37:     else return $1 + \max_{c \in b.children} \text{depth}(c)$
```

Algorithm 3 depicts the Bitcoin-specific pseudocode that includes its consensus protocol to decide on a particular block at some index (lines 27–37) and the choice of parameter m (line 25) explained in Section 2.5.2. When a fork occurs, the Bitcoin protocol resolves it by selecting the deepest branch as the main branch (lines 27–34) by iteratively selecting the root of the deepest subtree (line 30). 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 19–24 of Alg. 2; only the parameter m used in these lines differs between the Bitcoin consensus algorithm (Alg. 3, line 25) and this variant of the Ethereum consensus algorithm (Alg. 4, line 25).

2.7 The 51% Attack

Double spending is not as simple as we discussed in Chapter 1 when the external action is taken after one of the two transactions is committed by the blockchain. Rosenfeld's attack [Ros12] 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 attacker mining power and the number m of blocks the merchant waits before taking an external action. 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 expected to be 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 [ES14] 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 [DW13, PSS16, SZ15, GKKT16, NKMS16]. Decker and Wattenhoffer already observed that Bitcoin suffered from block propagation delays [DW13]. Godel et al. [GKKT16] analyzed the effect of propagation delays on Bitcoin using a Markov process. Garay et al. [GKL15] investigated Bitcoin in the synchronous communication setting. Pass et al. [PSS16] 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. 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.

Although its implementation differs, Ethereum initially proposed to use a variant of the GHOST protocol to cope with this issue [SZ15]. The idea was 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.

2.8 The GHOST protocol

As opposed to the Bitcoin protocol, Ethereum generates one block every $\sim$14 seconds. While it reduces latency and improves throughput (i.e., number of transactions committed per unit of time), it also favors transient forks as miners are more likely to propose new blocks without having heard yet about the

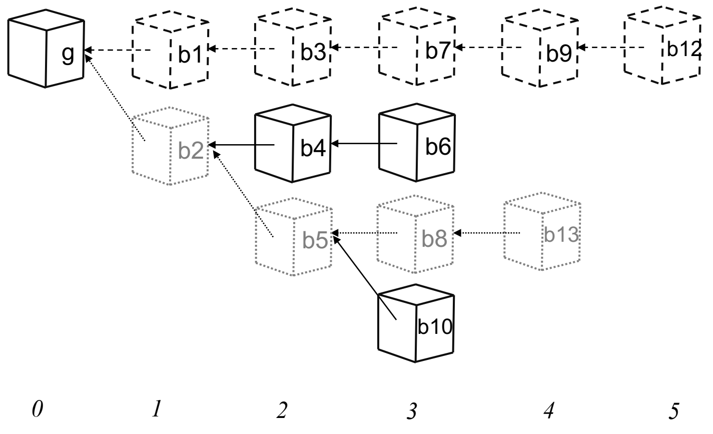

Figure 2.2: Nakamoto's consensus protocol at the heart of Bitcoin selects the main branch as the deepest branch (in dashed lines) whereas the GHOST consensus protocol follows the heaviest subtree (in dotted lines)

Algorithm 4 The additional field and functions used by the GHOST consensus algorithm at p_i

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

27: get-main-branch$()_i$:
28:     $b \leftarrow$ genesis-block$(B_i)$
29:     while $b.children \neq \emptyset$ do
30:         $block \leftarrow \text{argmax}_{c \in b.children}\{\text{num-desc}(c)\}$
31:         $B \leftarrow B \cup \{block\}$
32:         $P \leftarrow P \cup \{\langle block, b \rangle\}$
33:         $b \leftarrow block$
34:     return $\langle B, P \rangle$.

35: num-desc$(b)_i$:
36:     if $b.children = \emptyset$ then return $1$
37:     else return $1 + \sum_{c \in b.children}$ num-desc$(c)$
```

latest mined blocks. To avoid wasting large mining efforts while resolving forks, Ethereum suggested to use a variant of the 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 4). 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.

The main difference between Nakamoto and the GHOST consensus protocol is depicted in Figure 2.2, where the dashed blocks represent the main branch selected by Nakamoto's consensus protocol and the dotted blocks represent the main branch selected by GHOST.

2.9 Conclusion

Mainstream blockchain systems like Bitcoin and Ethereum require miners to solve a crypto-puzzle whose solution gets included as a proof-of-work in a new block. These blockchains recover from forks by running a consensus protocol that eventually chooses one branch among multiple ones, based on its length, difficulty or weight. By acquiring more than half of the mining power, an adversary can still double spend, an attack called the 51% attack. The next chapter is dedicated to explaining how the literature on consensus algorithms can help remedy this problem by avoiding forks.

2.10 Bibliographic notes

The impossibility to solve consensus with $f \geq n/3$ failures is due to Pease, Shostak and Lamport [PSL80]. A didactic explanation is presented in [Ray18].

The impossibility to solve consensus in an asynchronous system when there is at least one failure is due to Fischer, Lynch and Patterson [FLP85].

The idea of randomized consensus based on a common coin is due to Rabin [Rab83] whereas the one based on local coins is due to Ben-Or [BO83]. Some recent algorithms converge in $O(n^{2.5})$ expected time when based on local coins [KS16] and in constant expected time when based on a common coin [MMR15a]. Note that consensus has also been solved probabilistically without using randomization and with $f < n/3$ of Byzantine nodes by making the probabilistic assumption of a fair scheduler [BT83]. Sometimes both a probabilistic assumption and randomization are needed to solve consensus [TG19]. Partial synchrony is due to Dwork, Lynch and Stockmeyer [DLS88].

A replicated state machine shares similarity with blockchains [XPZ+16] and some interesting differences have been outlined [CGLR18]. The 51% attack was presented by Rosenfeld [Ros12] as a situation where an attacker exploits more than half of the mining power of the system to create a longer branch that can override other transactions. The attacker issues a transaction to a merchant and 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 depends on the number m of blocks the merchant waits for 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*, is expected to be successful, regardless of the value m. It is unclear whether a transaction is "confirmed" after 1, 6, 12 or more block inclusions but the notion of *committed* transaction blocks was originally defined in [NG16b] as a transaction that would be followed by m blocks—this necessary number

m of confirmations is generally provided by the authors of the blockchain system [Pro20, But15, But16a] but participants may choose their own parameter for security and speed. RepuCoin [YKDE19] aims precisely at coping against 51% attack by combining a proof-of-reputation with a proof-of-work. In Bitcoin [Nak08], a quarter of the mining power appears to be enough in theory to incentivize participants to join a coalition whose cumulative mining power will eventually reach strictly more than half of the total mining power [ES14], however, we are not aware of such an attack in practice. As we will see in Chapter 4, to attack blockchains without a significant mining power, researchers attacked the network.

Bibliography

[Bla02] Adam Black. Hashcash - a denial of service counter-measure. Technical report, Cypherspace, 2002. `http://www.hashcash.org/papers/hashcash.pdf`.

[BO83] 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, 1983.

[BPS16] Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. Technical Report 919, IACR Cryptology ePrint Archive, 2016.

[BT83] Gabriel Bracha and Sam Toueg. Asynchronous consensus and byzantine protocols in faulty environments. Technical Report TR83-559, Cornell University, 1983.

[But15] Vitalik Buterin. On slow and fast block times, 9 2015. https://blog.ethereum.org/2015/09/14/on-slow-and-fast-block-times/.

[But16] Vitalik Buterin. How should i handle blockchain forks in my dapp?, 1 2016. https://ethereum.stackexchange.com/questions/183/how-should-i-handle-blockchain-forks-in-my-dapp/203/#203.

[CGG19] Pierre Civit, Seth Gilbert, and Vincent Gramoli. Polygraph: Accountable byzantine consensus. In *Workshop on Verification of Distributed Systems (VDS'19)*, Jun 2019.

[CGLR18] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. DBFT: Efficient leaderless byzantine consensus and its applications to blockchains. In *Proceedings of the 17th IEEE International Symposium on Network Computing and Applications (NCA'18)*, 2018.

[CHT96] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, July 1996.

[CT96] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, April 1988.

[DW13] 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.

[EGSvR16] Ittay Eyal, Adem Efe Gencer, Emin Gün 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.

[ES14] 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.

[FLP85] 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.

[GKKT16] 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.

[GKL15] 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.

[GKW$^+$16] 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.

[KS16] Valerie King and Jared Saia. Byzantine agreement in expected polynomial time. *J. ACM*, 63(2):13, 2016.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[MMR15] 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), 2015.

[Nak08] Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system, 2008. http://www.bitcoin.org.

[NG16] 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.

[NKMS16] 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.

[Pro20] Bitcoin Project. Some things you need to know, 2020. https://bitcoin.org/en/you-need-to-know.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, April 1980.

[PSS16] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. Technical Report 454, Crytology ePrint Archive, 2016.

[Rab83] Michael O. Rabin. Randomized byzantine generals. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, SFCS '83, pages 403–409, 1983.

[Ray18] Michel Raynal. *Fault-Tolerant Message-Passing Distributed Systems - An Algorithmic Approach*. Springer, 2018.

[Ros12] Meni Rosenfeld. Analysis of hashrate-based double-spending, 2012.

[SZ15] 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.

[TG19] Pierre Tholoniat and Vincent Gramoli. Formal verification of blockchain Byzantine fault tolerance. In *6th Workshop on Formal Reasoning in Distributed Algorithms (FRIDA'19)*, Oct 2019.

[Woo15] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2015. Yellow paper.

[XPZ$^+$16] 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.

[YKDE19] J. Yu, D. Kozhaya, J. Decouchant, and P. Esteves-Verissimo. Repucoin: Your reputation is your power. *IEEE Transactions on Computers*, 68(8):1225–1237, 2019.