Frankfurt School Blockchain Center

MAY 2017

FSBC Working Paper

Consensus Methods in Blockchain Systems

Julian Debus

Distributed consensus systems are a novel means of establishing consensus between multiple parties concerning some piece of information, such as ownership of an asset. Consensus is recorded by cryptographically signing data, thus proving its authenticity. Blocks of signed data can be linked to previous blocks, thereby enforcing a temporal chain of blocks, giving rise to the term "blockchain". Since the emergence of Bitcoin in 2008, various blockchain systems have been proposed with very divergent properties and areas of focus. With this paper, we analyze properties and limitations of the most important classes of blockchain systems, with a specific focus on how these systems establish consensus. Distributed consensus has become a highly dynamic field with much work being committed outside of traditional academia. Accordingly, many aspects of distributed consensus lack formal analysis and concrete criteria under which newly proposed systems can be analyzed.

We propose four criteria under which the effectiveness of distributed consensus methods can be analyzed under economic considerations. Based on these criteria, we show how the incentives of economic agents in a consensus system accomplish a stable consensus state. Additionally, we are able to use these criteria to derive challenges in proof-of-stake consensus.

This working paper is an excerpt from a Master's Thesis entitled "Applicability and Weaknesses of Blockchain-based Consensus Methods in Financial Markets". **Julian Debus** has been student at the Frankfurt School of Finance & Management. He has been supervised by **Prof. Dr. Peter Roßbach**, Professor of General Business Administration and Business Informatics.

Frankfurt School Blockchain Center
www.fs-blockchain.de
contact@fs-blockchain.de

Follow us
www.twitter.com/fsblockchain
www.facebook.de/fsblockchain

Frankfurt School of Finance & Management gGmbH
Sonnemannstrasse 9-11
60314 Frankfurt am Main
Germany

Contents

1. Introduction

Distributed Consensus Systems are a novel means of establishing consensus between multiple parties about some piece of information, such as ownership of an asset. Consensus is recorded by cryptographically signing data, thus proving its authenticity. Blocks of signed data can be linked to previous such blocks, thereby enforcing a temporal chain of blocks, giving rise to the term *blockchain*. Since the emergence of Bitcoin in 2008, various blockchain systems have been proposed with very divergent properties and points of focus. Here we analyze properties and limitations of the most important classes of blockchain systems, with a specific focus on how these systems establish consensus.
The following work is structured as follows:

- Chapter 2 provides an overview of theoretical and technical building blocks of distributed consensus systems;
- Chapter 3 formally specifies distributed consensus systems and presents important classes of such systems;
- Chapter 4 introduces an economic model of agents in a consensus system;
- Chapter 5 analyzes properties of consensus systems and presents attacks on these systems;
- Chapter 6 closes with a discussion on the impact of distributed consensus systems on financial markets, and highlights select issues which may affect the practical implementation of consensus systems.

2. Distributed Systems & Cryptography

2.1. Asymmetric Cryptography

Asymmetric Cryptography is a cryptographic method which utilizes a pair of keys $\langle s, p \rangle$ to encrypt and decrypt information [40]. The *private* key s, and the *public* key p have to fulfill the following property in order to be a valid key pair: A piece of secret information which was encrypted with any of the two keys can be decrypted with the other, complementary, key.

More formally, encryption and decryption with the two keys is commutative: Let $\xrightarrow{k}$ denote the act of en-/decrypting with key k. Then, $\xrightarrow{p}\xrightarrow{s} = \xrightarrow{s}\xrightarrow{p} = I$, where I is the identity function.

We highlight `RSA` [47], which is based on modular arithmetic on large prime numbers, as well as `ECDSA` [28], which is based on the study of the algebraic structure of elliptic curves, as two prominent asymmetric cryptosystems.

2.2. Cryptographic Hash Functions

A cryptographic hash function is a mapping from a domain of arbitrary size (e.g. $\mathbb{N}^+$) to a codomain of fixed size (e.g. $\{x \in \mathbb{N}^+ | x < 2^{256}\}$) – a hash thus always establishes a many-to-one mapping (see figure 2.1). The result of applying a hash function to a piece of data is commonly referred to as *hash*. Important properties of cryptographic hash functions are [40]:

1. Preimage Resistance: It is computationally infeasible to find an input which hashes to a specific output.
2. 2nd Preimage Resistance: Given an input which hashes to a certain output, it is computationally infeasible to find a second input which hashes to the same output.
3. Collision Resistance: It is computationally infeasible to find any two inputs which hash to the same output.

The complexity of finding a preimage which is mapped to a given hash grows linearly with the cardinality of the hash function's codomain. A hash that was produced by a hash function fulfilling properties 1–3 can thus serve as a compact proof of the hashed input's existence, given that the function's codomain is sufficiently large.

We highlight `SHA-2` as a prominent family of hash functions [53]. Two widely-used `SHA-2` hash functions are `SHA-256` and `SHA-512`, with codomains of 256 and 512 bits, respectively.

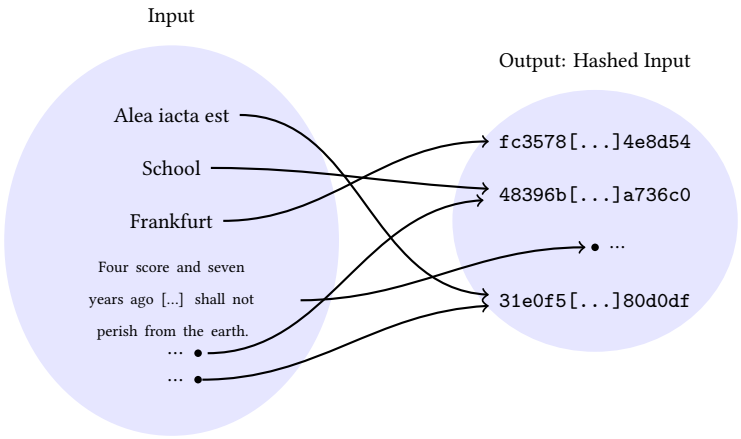


Figure 2.1.: Illustration of a Hash-Mapping

2.3. Electronic Signatures

Electronic signatures help establish the *authenticity* of a given piece of information. To this end, asymmetric cryptography and cryptographic hash functions are combined: To prove a document's authenticity, its author can calculate the document's hash using an appropriate hash function. This, in and of itself, does not prove the document's authenticity: Anyone might have calculated its hash. In order to uniquely link her identity to the document, the author encrypts the document's hash with a private key of hers, yielding an electronic signature. A document which is published along with an electronic signature is said to have been *signed*.

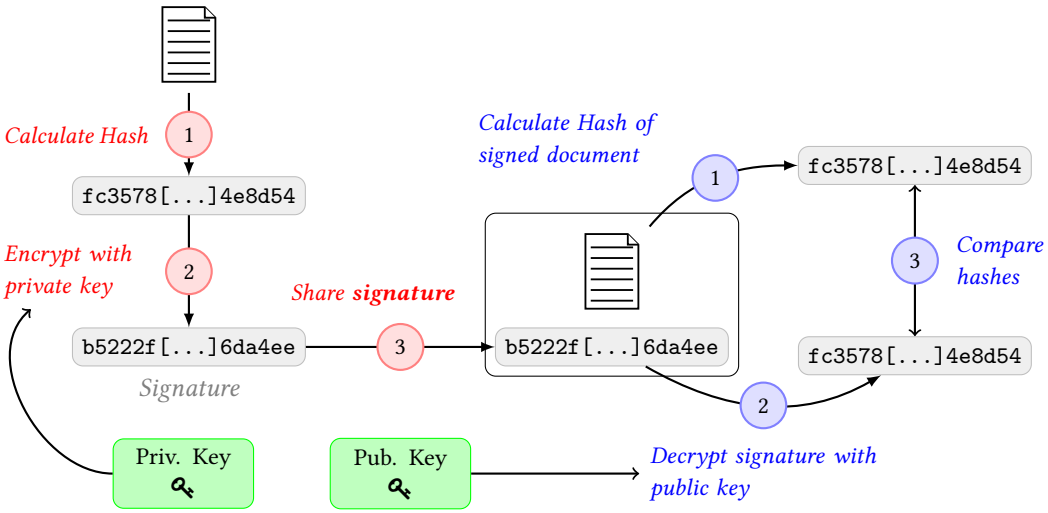


Figure 2.2.: Illustration of the creation of an electronic signature (red) and a subsequent verification of the signed document's authenticity (blue)

In order to verify the authenticity of a signed document, we can decrypt its electronic signature

with the assumed author's public key. Calculating the document's hash and comparing it with the decrypted hash then establishes the signee's identity: If the two hash values match, we can be sufficiently sure that the document was signed by a person in control of the complementary private key to the used public key. Figure 2.2 outlines the key steps in the creation and verification of an electronic signature.

2.4. State Machines

Abstractly, a state machine is a system which stores information, known as the system's *state*, and can deterministically change its state, also referred to as *transitioning* from one state to another, following some triggering event.
To define a state machine, one needs to specify (compare, e.g. [29]):

1. the set of admissible states $\mathcal{S}$
2. the set of inputs $\mathcal{I}$
3. a state-transition function which maps the existing state and provided input to some new state $\mathcal{I} \times \mathcal{S} \mapsto \mathcal{S}$
4. an initial state $s \in \mathcal{S}$

The concept of a state machine is fairly broad and can be applied to describe a wide range of systems. As an example, we illustrate how a ledger can be modeled as a state machine in figure 2.3.

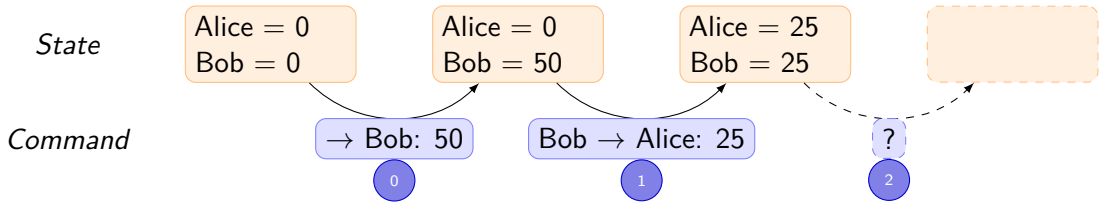


Figure 2.3.: Example: Ledger modeled as a state machine. After the second state-transition, the system's third state depends on the next command to be executed (2).

In the depicted example, the set of admissible states might be a number of accounts with associated balances. An input might be any of the three commands: Deposit(account, amount), Transfer(fromAccount, toAccount, amount), Withdraw(account, amount). Executing a command manipulates the state in a pre-specified manner, defined through the function mapping a given state to a new state upon the execution of a command. The initial state of the system might be an empty set of accounts.

2.5. Distributed Systems

In discussing consensus mechanisms in distributed systems, we consider the following definition of a distributed system [17]:

"A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages. This definition leads to the following especially significant characteristics of distributed systems: concurrency of components, lack of a global clock and independent failures of components."

3. Distributed Consensus Systems

In the following, we provide an overview of distributed consensus systems. We start this review, for historical reasons, with a brief description of *Bitcoin* [42]. This facilitates an analysis of other systems which build and extend on Bitcoin's innovations.

3.1. Bitcoin: A Distributed Ledger System

In 2008, Nakamoto laid the theoretical foundation for modern consensus systems in the landmark paper "Bitcoin: A Peer-to-Peer Electronic Cash System" [42]. Here, we briefly summarize the main features of Bitcoin.

Transactions involving electronic coins We define an electronic coin as a chain of digital signatures. In order to transfer ownership of a coin, its old owner cryptographically signs a hash of the current coin along with the new owner's public key (see figure 3.1). This signature explicitly shows to the outside world that the old owner has passed ownership of their asset to the new owner. We note that a coin is thus simply a cryptographically linked transaction history, documenting its ownership across time. In figure 3.1, for example, Owner 3 is currently in possession of the depicted electronic coin. The history of ownership can be traced by recursively verifying the authenticity of the linked signatures, from owner 2's signature, to owner 1's signature, ..., up to the coin's first owner.

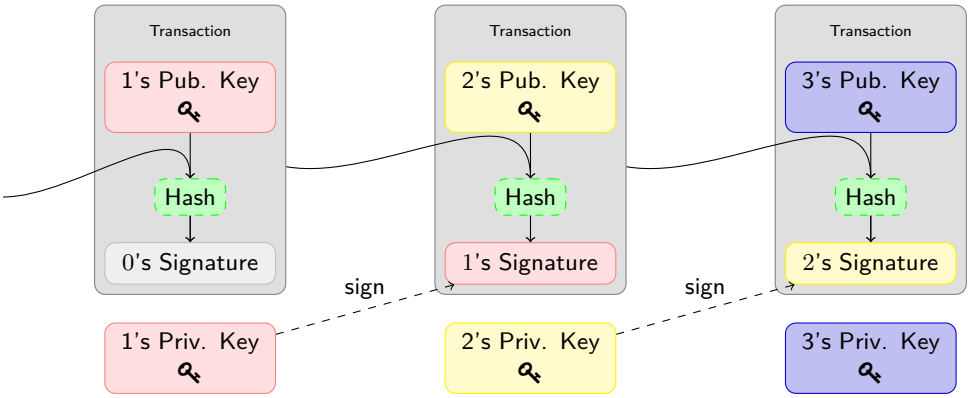


Figure 3.1.: Chain of cryptographic signatures, proving ownership of an electronic coin, adapted from [42]

Distributed Timestamping To make this model of an electronic coin work, we need to ensure that no fork in its history can create an ambiguous situation where more than one key pair could claim ownership of the coin, referred to as the "double-spending problem". Figure 3.2 outlines the nature of this problem: Owner *1* transfers a coin of hers *twice* — first to Owner *2a*, then to Owner *2b*. Without any other measures, both *2a* and *2b* might reasonably expect to be legitimate owners of the depicted coin.

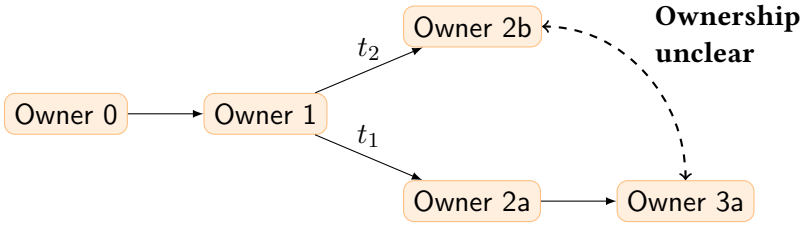


Figure 3.2.: The Double-Spending Problem: Fork in a coin's ownership history

A natural solution to this problem is to enforce a temporal ordering among transactions. If, for example, we knew that *1* transferred her coin to *2a* first, we could trivially invalidate the second transaction to *2b*.

In a centralized system, this setup can be implemented relatively trivially: Participants send proposed transactions to a centralized party C which regularly publishes collected transactions to the outside world, along with the publication time. C's function in the system is that of a timestamping server. In doing so, C ensures that a recent representation of the transaction set in the centralized system is available to all participants. The situation depicted in figure 3.2 can thus be resolved by disregarding the latter (as determined by the respective timestamp) of the two transactions ($1 \rightarrow 2b$).

A decentralized system can, by definition, have no central party C. This gives rise to a *consensus problem* which we specify more formally in the following sections: Participants in the decentralized system need to coordinate among each other and agree on some temporal ordering of transactions in the system. One of Bitcoin's main contributions is a solution to this problem by means of a distributed timestamping service that is backed by a novel consensus method which has, so far, proven effective in a distributed setting.

Bitcoin requires its participants to collect sets of new transactions into a "block". Each participant then tries to publish her newly formed block by solving a computationally hard puzzle: Searching for a value ("nonce") that, combined with the previous block's hash, and all collected transactions, hashes to a value within a certain subspace of the hash function's codomain. This type of a partial preimage search using hash functions was first proposed by Back in 2002 in his work on *Hashcash* [2]. The fact that a block references a previous block's hash gives rise to a linked list-type structure of blocks which we call the *blockchain* (see figure 3.3).

As soon as a block with an appropriate nonce is found which solves the puzzle, the block is published to all participants. Each participant can verify the solution's correctness easily by verifying that the published block's hash is within the mandated subspace. The only way for a participant to alter the published state of transactions would be to solve a new puzzle

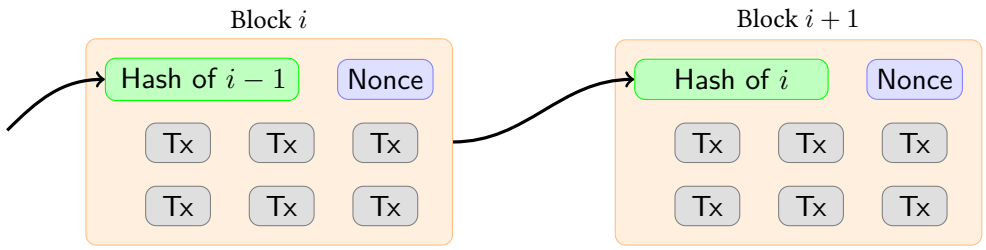


Figure 3.3.: The Bitcoin Blockchain, adapted from [42]

with a new set of transactions, thus creating a fork in the blockchain. In light of such forks, the set of transactions in the longest chain of blocks is by convention regarded as valid, while all other transactions are ignored. As more and more blocks are added to extend the blockchain, it becomes increasingly infeasible for a fork in the chain's history to "catch up" with and overtake the longest chain [42]. Thus, the overall system is secure as long as well-intended participants are the ones adding blocks to the chain the fastest. This is ensured as long as well-intended participants control the majority of computing power in the network: A chain of invalid blocks could not overtake a faster-growing chain of valid blocks over the long-run. We recognize that Bitcoin's security thus strongly depends on the total computing power allocated (by well-intended participants) towards publishing new blocks.

3.2. Implementation of Distributed Ledgers with Distributed State Machines

Abstractly, the Bitcoin network maintains and updates a distributed ledger of the participants' accounts in the system. In this function, we can understand the network as an implementation of a distributed state-machine where account balances represent the state that is being changed in accordance with executed commands on the state-machine. Since knowledge of the ordered set of accepted transactions is sufficient to infer the balance of each account, we will henceforth refer to the set of accepted transactions in the system as the (state-machine) *state*.

To facilitate the discussion in further sections, we briefly formalize the notion of the blockchain as a means for applying and tracking changes to a state. A blockchain forms a directed acyclic graph of changes. We can thus describe a blockchain $C = (\mathcal{B}, \mathcal{E})$ as consisting of a set of changes $\mathcal{B}$ that are connected through a set of directed edges $\mathcal{E}$ which themselves are ordered 2-subsets of $\mathcal{B}$ and do not give rise to any cycles. By means of an example, $C = (\{0,1,2,3,4\}, \{\{0,1\},\{1,2\},\{2,3\},\{1,4\}\})$ corresponds to the graph depicted in figure 3.4.

For any edge $\{a,b\}$ of changes a and b, we say that b is *depending* on a or, equivalently, that a *supports* b. A blockchain always starts from a unique *genesis* state, which is the only member of $\mathcal{C}$ which is not depending on other changes. We define the *height* h of a change as the number of edges between the change and the genesis state. In figure 3.4, for example, $h(4) = 2$, $h(0) = 0$. We furthermore introduce the notion of a *tip* of a chain of changes. A tip is a change which does not support any other changes in the blockchain. In a blockchain with more than two tips, we call the subsets of changes and edges between a tip and the genesis state

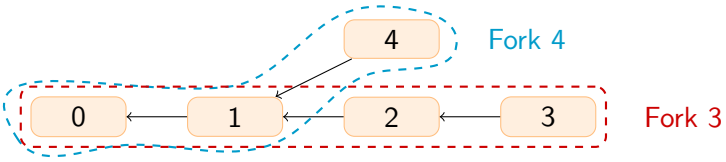


Figure 3.4.: Example of a blockchain structure with two forks

forks of the blockchain, and the blockchain itself *forked*.

In keeping with the view of a distributed state-machine, the set of changes corresponds to commands which are executed in the specified order. In this context, we can understand the distributed timestamping scheme in Bitcoin as a method for participants in a network to agree on the next command that is to be executed on a state machine machine, thereby updating accounts.

3.3. Distributed Consensus Methods

We define a distributed consensus method as a means of securely choosing a command that is to be executed on a distributed state-machine. In examining Bitcoin, we have briefly reviewed one example of such a consensus method which is used to securely update account balances. Since Bitcoin's emergence in 2008, numerous other consensus methods have been proposed that can be applied to the problem of updating distributed state in a blockchain.
We let $\mathcal{C}$ be the set of all admissible blockchains and define a blockchain-based consensus method as consisting of

- *Consensus-Determination Function* $\mathcal{D}$: Determines the consensus state (the ultimate block in a chain of blocks) from any given blockchain structure; $\mathcal{D} : \mathcal{C} \to \mathcal{B}$.
- *Consensus Strategy* $\mathcal{S}$: Determines which block a client will apply next to a given blockchain structure; $\mathcal{S} : \mathcal{C} \to \mathcal{B}$.
- *State-Transition Function* $\mathcal{T}$: Governs how a block's application to a blockchain transforms the blockchain; $\mathcal{T} : \mathcal{C} \times \mathcal{B} \to \mathcal{C}$

In order to allow for a more structured analysis of these various methods, this next section discusses consensus methods by grouping them into classes centered around the notion of "economic sets". Economic sets represent various means by which "the right to perform state transitions" [11] is distributed across participants in a distributed system. We thus cluster consensus methods by their underlying consensus strategy $\mathcal{S}$. To improve legibility, we will henceforth refer to the likelihood of a given participant "winning" the right to publish a state-transitioning block (based on an associated member in an economic set) as her *state-transition probability*.

3.3.1. Computational Power: Proof of Work

The central idea behind *proof-of-work* systems is to mandate participants, wishing to publish a state transition, to work on a problem that requires (on average) some amount of computational work to be solved. As soon as participants discover a solution to a given problem, they publish their proposed state transition along with this solution. The key insight to understanding how consensus is established in these system is:

- Any number of participants might publish proposals for state transitions. In order to know which state transition to accept, participants continuously verify the validity of the solutions that are published along with state transition proposals.
- Participants accept a state transition as soon as a solution has been validated, since this signifies that someone worked on and solved the problem. The problem's solution thus serves as a *proof of computational work* that was performed by the publisher.
- All participants can thus come to consensus about the work that was performed in trying to publish a state transition. Considering that the work that needs to be performed has an economic cost (usually in electricity), the publisher effectively incurs cost to be able to publish a state transition.
- As long as the majority of computing power in the system is controlled by honest participants who try to publish valid state transitions, one can expect the system's state to be valid over the long-term. This is due to the fact that honest participants can expect to "win" state transitions most frequently, forming the longest chain. Any fraudulent state transitions published on other chains, which still carry an economic cost, would be invalidated by the longest chain. As such, participants can confidently come to consensus about the application of a state transition as soon as a small set of proofs of work has been published.

In the described setting, the economic set $\mathbb{P} = \{(a_1, p_1), (a_2, p_2), \ldots, (a_n, p_N)\}$ of computational power p (measured in number of computations per unit of time) controlled by account a governs the state-transition probability of any a. More concretely, the state-transition probability f of an account a is monotonically increasing in the computing power p controlled by a.

$$f(p_1) \leq f(p_1 + \epsilon)$$

Proof-of-Work Puzzle The main requirements of any puzzle that is to be used in a proof-of-work setting are two-fold: (1) Its difficulty should ideally be almost arbitrarily scalable, so as to ensure that the proof of work does not become trivially easy to obtain with advancements in computing power over time; (2) a found solution should be easily verifiable, so as to keep verification cost in the network small. A formal specification of admissible puzzles is an area of active research [1].

[1] Preliminary results define a class of puzzles called "scratch-off"-puzzles [41]. Their key characteristic is that the process of finding a solution comprises many independent attempts of performing some task. This property

To illustrate the differences in potential proof-of-work puzzles, we review the puzzles employed in Bitcoin [42], as well as Ethereum [56], two consensus systems with the highest market value of their embedded monetary supply (see appendix B).

Bitcoin employs a partial preimage search for its proof-of-work puzzle: The concrete problem to be solved is finding a block (containing transactions, the previous block, as well as a nonce) whose hash starts with d zero-bits. In this context, d is understood to be the difficulty of solving a given puzzle.

$$\texttt{SHA-256}(\text{Transactions, Prev. Hash, Nonce}) \stackrel{!}{=} 0\text{b}\underbrace{0000\ldots000}_{d\text{ 0-bits}}\ \underbrace{xx\ldots xxxx}_{256-d\text{ bits (0 or 1)}}$$

$$\Leftrightarrow \texttt{SHA-256}(\text{Transactions, Prev. Hash, Nonce}) \leq 2^{256-d}$$

Assuming equally distributed hashes, the time complexity of solving a given puzzle is thus $\mathcal{O}(2^d)$[2]. The difficulty is determined by "a moving average targeting an average number of blocks per hour. If they are generated too fast, the difficulty increases." [42]. The standard Bitcoin implementation targets the generation of a block every 10 minutes. This mechanism controls the effects of advances in microprocessor architectures (Moore's Law[3]), as well as the number of participants in the network, on the rate with which blocks are published. Computing a `SHA-256`-Hash is mainly a CPU-bound problem, thus making the overall puzzle CPU-bound.

Ethereum proposes "Ethash[4]" as its proof-of-work puzzle. Ethash is designed to be memory-I/O-bound. The authors argue that this ensures that state transition probabilities in the system are more equally distributed, because commodity memory operates closer to optimal state-of-the thresholds than do commodity CPUs.

Ethash is executed in two steps. Firstly, a large data structure, consisting of 1GB of 64-byte fields, is computed and written to memory. Each 64-byte field in turn depends on a small number of items in a smaller cache, which can be computed from a seed that depends on the block history. Secondly, a nonce is chosen from which 128 fields (pseudo-randomly chosen) are combined and hashed using a combination of the `FNV` [24] and `SHA3` [52] hash functions. The fact that 128 fields are randomly chosen from the data set forces many non-sequential memory accesses, thereby bounding the execution time not by the computation time of the hash function, but by the retrieval time of fields from memory.

The objective in Ethash is, similar to Bitcoin, finding a partial preimage to the proposed hashing scheme by finding a nonce which selects fields which produce a hash in the subspace defined by a difficulty d. We can thus understand Ethash's proof-of-work puzzle as a modified

ensures that the probability of solving a puzzle scales linearly with the amount of computational work. Depending on the exact type of computational tasks that are performed and the configuration of the computer executing said computations, the speed of execution can be bound, among others, by the processing power of the CPU (CPU-bound problems) or the speed with which data can be stored or retrieved from memory (memory-I/O-bound problems).

[2] For $d = 0$, any hash is a valid solution; for $d = 3$, only an eighth of all hashes are valid solutions, ...

[3] Prediction by Gordon Moore in 1965: "The performance of microprocessors is set to double every two years." (paraphrased). Famously, this prediction has largely held true to the present day.

[4] `github.com/ethereum/wiki/wiki/Ethash`

version of Bitcoin's puzzle where the pseudo-random field retrieval is added as a layer of indirection that binds the execution speed of the puzzle to the speed of memory-I/O operations.

Other Proof-of-Work Systems Apart from a search in `SHA-256` or `SHA-3`-space, many other puzzles have been conceived and deployed in proof-of-work systems. We group these puzzles loosely into two classes: One which introduces another hash function, such as `scrypt` [44], and one which introduces an altogether different puzzle to be solved, such as the search for special prime numbers [31].

Incentivization As noted earlier, obtaining a proof of work carries a real economic cost, mostly in electricity. As such, proof-of-work systems cannot function without economic incentives for participants to expend the cost associated with proof-of-work. A common method to introduce incentives for providing a proof of work is to award the publisher of the proof with newly-created coins from the embedded monetary supply of the system. In light of the similarities to mining for gold — spending resources to acquire something valuable and adding it to global supply — the process of trying to find a proof of work is referred to as "mining".
This process has three important effects:

1. It incentivizes participants to spend resources on mining. The total amount that is spent on mining will be — in an idealized setting — as high as the value of the issued coins.

2. It incentivizes participants to work on extending the longest chain of blocks. Since the account ledger is — by convention — only derived from the longest chain, any coins awarded on shorter chains are considered invalid and thus worthless. This serves as a strong disincentive to working on shorter forks in the blockchain and leads to a convergence of the network's computing power on the longest chain.

3. It introduces a form of inflation, as the monetary supply in the system grows continuously. The cost of this inflation is borne by other participants in the system. The overall system can thus be understood as to be paying for the electricity that is used to secure the system and advance the system state.

The concrete incentivization schemes in different systems mainly differ in the temporal development of the rewards for publishing a proof of work.

Bitcoin The current standard implementation of Bitcoin mandates that the reward for publishing a new block be halved roughly every 4 years, starting from 50 coins in 2009, to 25 coins in 2013, to zero coins in roughly 2140. At a current[5] BTC-EUR exchange rate of about 961, this incentivization scheme sets out a reward of roughly € 12k per published block. At a publication rate of approximately six blocks per hour, the total daily rewards paid to miners in the Bitcoin network are thus in the range of € 1.4m. Induced by these rewards, growing demand for Bitcoin, as well as advances in computing technology, the hash rate, expressed in number of

[5] as of March 22, 2017; `exchange.coinbase.com/trade/BTC-EUR`

hash values calculated per unit of time, has grown tremendously to approximately $3.3 \cdot 10^{18}$ calculated hash/s in March of 2017 (see figure 3.5).

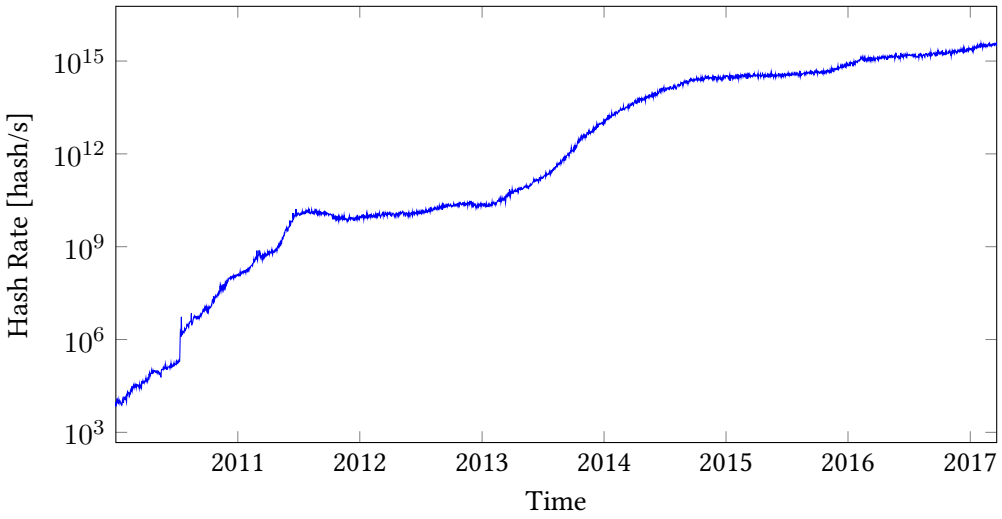


Figure 3.5.: Hash Rate [hash/s] of the Bitcoin network, Data Source: `bitcoin.info`

In order to put some perspective on the hash rate above, we present hash rates of a modern *Intel* CPU, as well as a microprocessor by *Bitmain*, specifically designed for the calculation of `SHA-256` hashes in table 3.1.

Table 3.1.: Hash Rates of select microprocessor architectures

Microprocessor	`SHA-256`-Hash Rate [10^6 Hash/s]	Price [USD]
Intel Core i7 5820K[6]	11	430
Bitmain AntMiner S9[7]	14,000	2100

Other Proof-of-Work Systems Whereas Bitcoin proposes rewards which decay at an exponential rate, other systems set out constant rewards, thereby adding a fixed amount to the monetary supply of the given system. Freicoin[8] is a cryptocurrency which uses constant inflation, combined with demurrage fees on existing holders (as an alternative to inflation). The idea behind this structure is to keep the monetary supply (and thus prices) stable, while simultaneously disincentivizing the hoarding of balances which can be expected in low-inflation currencies.

[8] `http://freico.in/about/`

3.3.2. System Stake: Proof of Stake

The security of proof-of-work systems directly depends on the amount of computational work expended. The growing hash rate in the Bitcoin network, for example, makes attacks on the network costly. This security comes with significant economic cost: It is estimated that — in 2013 — the amount of energy allocated towards bitcoin mining (in terms of electricity cost for the operation of CPUs and cooling systems) equaled that of the country Ireland [21].

In light of the cost associated with *proof-of-work* consensus, proposals have been brought forth for a consensus mechanism centered around a different economic set. *Proof-of-stake* systems distribute state transition rights, among others, according to existing balances held by an account (the "stake" in the system) [10, 11, 32, 34]. Let $\mathbb{B} = \{(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_N)\}$ be the economic set of balances b held by an account a in a system.

One option is to distribute the likelihood $f \in [0, 1]$ of a given account publishing a state transition per unit of time purely according to its associated balance. The likelihood f could, for example, be modeled as a monotonically increasing function in the account balance.

$$f(b_1) \leq f(b_1 + \epsilon)$$

This could correspond to the following puzzle [10]:

$$\texttt{SHA-256}(\text{Acc. Address}, \text{Prev. Hash}, \text{Timestamp}) \leq 2^{256} \cdot \frac{\text{Balance}}{d}$$

We note that this type of puzzle is, at its core, still based on the concepts of proof-of-work. The energy consumption of such a system would, however, be significantly lower, because the rate at which the hash space can be searched is limited: The number of hashes which can reasonably be calculated per unit of time depends on the resolution of the timestamp, the only input to the hash function which changes. A timestamp-resolution of 1 second would, for example, limit the amount of hashes that can reasonably be calculated per second to the total number of accounts multiplied with the number of potential previous hashes of blocks from which the blockchain might reasonably be extended.

A proposed modification to this puzzle incorporates the "coin age", the duration that a given coin has been held by an account, into the problem specification [10, 32]. An example of a system which distributes state transition probabilities according to an account's coin age is Peercoin [32]. A corresponding puzzle might be defined as follows:

$$\texttt{SHA-256}(\text{Acc. Address}, \text{Prev. Hash}, \text{Timestamp}) \leq 2^{256} \cdot \frac{\text{Balance} \cdot \text{Coin Age}}{d}$$

This setup has one inherent problem: Making the problem's difficulty inversely proportional to a respective account's aggregated coin-age-weighted balance can give rise to dispersion of state transition probabilities in the system. Since consensus systems usually incentivize the publication of a state transition with additional balances, one would expect holders of large balances to accumulate additional balances over time. Importantly, this effect would also act as a disincentive to transactions in the systems, since old balances carry more value than newly received balances. A solution to this problem is to have participants destroy accumulated coin age when publishing a new block[9] [32].

[9] This can, for example, be achieved by transferring coins to oneself, thereby "resetting" the coin age

In the past two years, variants of proof-of-stake systems have emerged which distribute state transition rights among a subset of all participants in the systems. This subset may be defined and selected for in a number of ways, for example by only considering participants who have posted a security deposit, or who are trusted by stakeholders in the system. These selection criteria function as a disincentive against malicious behavior, when malicious participants run the risk of having their security deposits seized, or having their state transition rights revoked.

The following section presents variants of general proof-of-stake which place state transition rights in the hands of a small set of participants. We present delegated proof-of-stake, which has all participants vote for the constituents in the set, as well as Casper, which distributes state transition probabilities equally among participants willing to put down a security deposit.

Delegated Proof of Stake As an evolution to the concept of proof-of-stake consensus, delegated proof-of-stake systems distribute state transition rights among a small set of participants, the delegates.

Delegates are elected for a period of time based on a variety of potential methods. These methods can be automated, for example by merely selecting the top X participants controlling stake in the system, or may require all participants in the system to "vote" for trustworthy delegates. Such votes can be understood as spanning a graph in the delegated proof-of-stake system, where participants are nodes connected by directed vertices weighted with the stake of the voting user. Voter election can then be based by tallying up the weights of incoming votes per participant and selecting the top X participants, or those participants owning X percent of the system stake votes. We note that the security of such a system relies — depending on the exact scheme — on the benevolence of the top stakeholders in the system, or the ability of those stakeholders to predict the benevolence of future delegates.

The first system to propose a delegated proof-of-stake mechanism is Bitshares[10] in 2014.

Ethereum: Casper The *Ethereum Foundation* is currently developing "Casper", a proof-of-stake system which spreads state transition probabilities across a set of *validators*, participants who have registered to become eligible to propose state transitions and have committed a fixed security deposit [14]. Validators in Casper are tasked with the creation of new state transition proposals, as well as the quotation of bets on the state of the system, with their security deposit serving as collateral for all bets.

Casper spreads state transition probabilities equally — that is, in proportion to the committed security deposit/system stake — among all validators. A new state transition proposer is chosen randomly every five seconds based on some measure of entropy in the system. One such measure which is being proposed is validator activity during the time of one state transition. The selected validator can then publish her proposal to all other validators.

Once a state transition is published, all validators bet on the system's likelihood of converging on a state transition. Based on some scoring rule [17, 29], which is currently specified as a linear combination of a quadratic and a logarithmic scoring rule, validators win or lose part of their security deposit in the subsequent state transition. Whether a validator wins or loses a bet is again determined by the state of the system after future state transitions have been

[10] `bitshares.org`

applied. Which state transitions are applied depends, in turn, on decisions by validators which are informed by system-implied probabilities of convergence on a state transition. We conclude that the system's dynamics is governed by a recursive economic game whereby participants bet on the future state of the system, thereby affecting the system's future state.

Incentivization The nature of proof-of-stake systems strongly aligns the incentives of those participants most likely to win state transitions. Considering that the value of any participant's stake in a proof-of-stake system directly depends on the system's security, those with the highest incentive (in terms of monetary value) to keep the system's state valid are the ones in charge of enforcing a properly running system.

When comparing proof-of-stake to proof-of-work, this (mis-)match between control of state transition rights and interest in system security becomes even more evident: In proof-of-work consensus, those participants who own the majority of computing power are not necessarily the ones with the highest incentives to keep the system state valid.

Proof-of-stake systems can distribute value (usually in form of inflation) to participants who publish a state transition as a further incentive, similar to proof-of-work systems. In light of the fact that proof-of-stake system do not require expenditure of external economic resources, however, the inflation invested in rewards for state transitions can be set much lower than that of proof-of-work systems which additionally need to compensate participants for incurred electricity cost.

3.3.3. Inter-Network Relationships: Byzantine Agreement

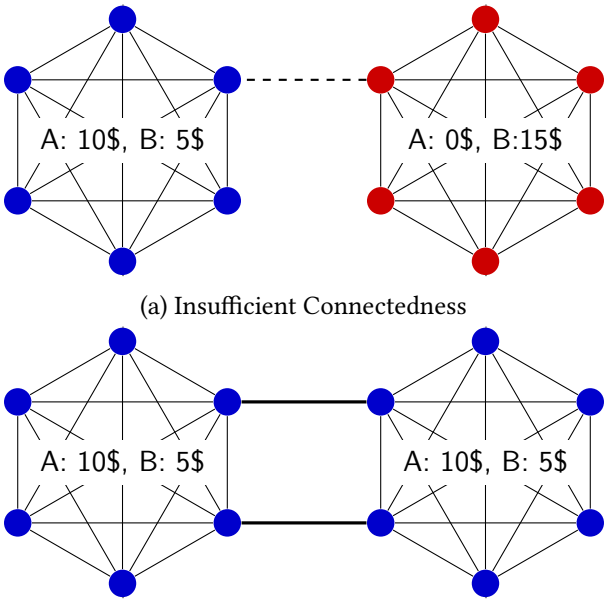
Another method for establishing consensus in a distributed setting is Byzantine Agreement. Byzantine Agreement comprises a class of systems that try to solve the *Byzantine General's Problem* [36], first described by Lamport, Shostak, and Pease, in which consensus has to be established in the face of arbitrary failures of participants. These "Byzantine" failures can include malicious actors making incorrect statements, as well as statements being lost, e.g. due to technical problems. Two prominent methods for establishing Byzantine Agreement are *Practical Byzantine Fault Tolerance* (PBFT) [15], as well as *Paxos* [35].

Byzantine Agreement is a particularly attractive method for finding consensus since it can be implemented such that consensus is reached both quickly and economically, when compared to *proof-of-work*. Moreover, it does not tie participation in the system to ownership of assets, as is the case in *proof-of-stake*. A key challenge in Byzantine Agreement systems is that they generally mandate the availability of a known, unique, and fixed set of participants who determine consensus. If this set is not strictly determined and shared by participants "attacks are always possible except under extreme and unrealistic assumptions of resource parity and coordination among entities" [20].

Probabilistic Voting (Ripple) A current application of Byzantine Agreement, focussing on advancing the state of a distributed blockchain ledger, is "The Ripple protocol consensus algorithm" (RPCA) [48]. The distributed ledger tracks the balance and currency of user accounts. Each participating node maintains a list of other trusted nodes on the network, the *Unique Node List* (UNL), and runs rounds of RPCA.

In a round of RPCA, each node collects valid transactions, which have not already been applied to the ledger, and shares them with the network in what is known as a *candidate set*. Nodes collect the candidate sets from nodes on their UNL and vote on the validity of all transactions received. In an iterative process, nodes then discard all transactions which have not been voted on by at least $x\%$ of the nodes in their UNL, where x strictly increases up to a level of 80%, and share and collect updated candidate sets. All transaction that are validated in the last iteration are considered to be validated and are applied to the ledger.

From the fact that $x = 80\%$, we infer that RPCA can withstand up to 20% of all nodes experiencing Byzantine failure, and still operate as desired. When the number of ill-behaving nodes exceeds 20% but is below 80%, consensus can be significantly impaired, with a worst-case behavior of all nodes consenting, in each round of RPCA, to not apply *any* transactions. In the case of 80% or more bad nodes, the system fails and can no longer make assertions about the correctness of the distributed ledger. As an additional requirement, the authors highlight the need for a minimum level of interconnectedness between nodes in order to ensure a consistent consensus across all nodes. Figure 3.6 illustrates this requirement: If a network is strongly clustered, e.g. such that two sub-graphs are connected through fewer than 20% of total nodes, each sub-graph can come to a different consensus (see figure 3.6a).



(a) Insufficient Connectedness

(b) Sufficient Connectedness

Figure 3.6.: Connectedness of nodes required to prevent a fork, adapted from [48]

From the point in time in which two parts of the network come to different consensus, the distributed ledger experiences a fork where two versions of the ledger coexist in the network. This suboptimal result imposes a strong restriction on the set of allowed UNLs under which RPCA performs as desired. If this restriction is violated and UNLs are picked without proper network-wide coordination, RPCA can fail and produce inconsistent ledger histories amongst participating nodes (Figure 3.6b). This has, in fact, already been empirically confirmed: In

December 2014 a version of RPCA with an improper set of UNLs experienced a ledger fork, creating a situation where the ownership of economic resources was unclear[11]. To resolve the forked network state, approximately seven hours worth of transactions had to be manually "replayed" on one chain. Additionally, the network was forcibly moved to a centralized consensus model with only one participant having state-transition rights.

Federated Byzantine Agreement (Stellar Consensus Protocol) Motivated by the aforementioned ledger fork in an implementation of RPCA, Mazières proposed Federated Byzantine Agreement (FBA) in November 2015 and furthermore made explicit the conditions which must hold in order for a system to deterministically reach consensus [39]. FBA relaxes the requirement for a centrally supervised list of UNLs and instead gives participants freedom in choosing their "trusted" nodes in what the author calls *quorum slices*. Each node in a Federated Byzantine Agreement System (FBAS) can specify various quorum slices, sets of nodes whose consensus on some fact is sufficient to also convince the node of said fact. The composite of all quorum slices forms a network of trust which gives rise to *quorums*. Quorums are sets of nodes which are sufficient to convince a subset of nodes in an FBAS of some fact. As such, a quorum is a weak superset of all participating nodes, generated by the union of (at least) one quorum slice of each participant. In this setting, an FBAS maintains consensus if two properties are met: *Quorum Intersection* and *Quorum availability*.

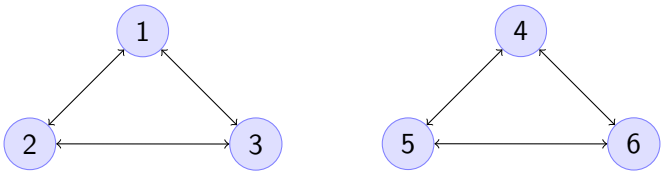
Quorum Intersection mandates that the intersection of all available quorums in an FBAS contain at least one correctly-functioning and honest node. Quorum intersection is required in order to prevent forks in the ledger history. Figure 3.7 outlines the requirement for quorum intersection: Figure 3.7a shows an FBAS with two quorums that do not intersect. It is impossible for {1,2,3} and {4,5,6} to reach consensus. Figure 3.7b shows an FBAS with two quorums that overlap at node 7. As long as node 7 behaves correctly, this system can come to consensus. If node 7 decides to act maliciously, however, it can convince of {1,2,3} and {4,5,6} of contradicting facts, causing the network to fail arbitrarily.

Quorum Availability mandates that the FBAS contain at least one quorum that does not contain incorrectly-functioning or malicious nodes. If Quorum Availability is violated, the FBAS can be in a stuck state where no consensus can be reached because of failing nodes that do not answer to requests.

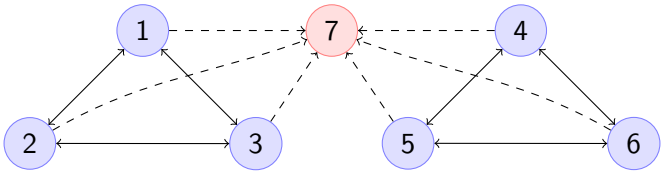
In summary, the correct behavior of an FBAS for any individual node solely depends on the choice of its quorum slices. The trade-off between quorum intersection and quorum availability that arises in choosing quorum slices is summarized by the author as follows [39]:

> "Nodes must balance [between quorum availability and quorum intersection] in slice selection. All else equal, bigger slices lead to bigger quorums with greater overlap, meaning fewer failed node sets [...] will undermine quorum intersection when deleted. On the other hand, bigger slices are more likely to contain failed nodes, endangering quorum availability."

[11] Stellar Development Foundation: Safety, liveness and fault tolerance — the consensus choices, `stellar.org/blog/safety_liveness_and_fault_tolerance_consensus_choice/`



(a) No Quorum Intersection



(b) Quorum Intersection at node 7

Figure 3.7.: Quorum intersection, adapted from [39]

3.3.4. Consensus Methods based on other Economic Sets

The following consensus methods have not yet seen widely used implementations. For the sake of completeness, we briefly present the concept behind each system.

Proof of Activity

A system based on proof-of-activity mandates two steps for the publication of a block [4]:

1. Participants try to solve a proof-of-work puzzle.
2. Once a valid block has been discovered, a set of past publishers of blocks is derived from the discovered block's nonce. In order to be successfully published, a block needs to be signed by all selected publishers.

The successful publication of a block thus requires cooperation between (a) participants who are controlling significant shares of computing power in the present, and (b) past publishers of blocks which will have had control over significant computing power in the past. As such, proof-of-activity systems are thought to be safer against attacks, since merely controlling the majority of computing power in the present is no longer sufficient to control the system's progress. This same fact, however, can cause major delays in a system's progress, if past publishers are no longer available to sign a new block.

Proof of Burn

A proof-of-burn systems distributes the state-transition probabilities in accordance with the number of coins which are "burned" by their owner. The process of burning a coin entails sending it to a randomly selected address which is unlikely to be under control of any participant[12]. The amount of burned coins then serves as an everlasting ticket in a lottery for the right to publish

[12]To illustrate this process, we documented the process of burning an amount of 0.0001 Bitcoin in appendix D

state transitions. Participants can then engage in a proof-of-work-style puzzle whose difficulty is inversely related to the amount of burned coins, similar to the concept presented in section 3.3.2. Slimcoin[13] is an example of a system which is based on proof-of-burn and was released in 2014.

Proof of Capacity

Proof-of-Capacity systems distribute state-transition probabilities in accordance with the memory and hard drive capacity that an account can read and write to. Burst[14] is an example of a proof-of-capacity system which requires participants to store and manipulate large amounts of random data. The frequency with which a new block is found is inversely proportional to the amount of data that was saved.

Proof of Stake Velocity

Proof-of-Stake-Velocity systems are a special subset of general proof-of-stake systems [46]. State transition probabilities in a proof-of-stake-velocity system are distributed according to the economic set of coin-age weighted balances. The coin-age of a given balance is calculated through a function which decays exponentially with increasing holding time. A newly acquired balance thus earns more coin-age than one that has not been transferred in a while. As such, this design induces participants to turn over their balances more quickly, thus giving an average balance a higher "velocity".

Proof of Bandwidth

Proof-of-Bandwidth systems allocate state transition probabilities according to the bandwidth that a participant makes available in a network [25]. To this end, participants are connected in "paths", sub-graphs within the larger network, which are connected through weighted edges. This weight is measured and signed by decentralized servers in the network in accordance with the throughput of information transfer over a path. This design induces participants to (a) commit bandwidth to a network and (b) connect with other participants which also have high bandwidth. Torcoin represents one of the first applications of this technology. It uses proof-of-bandwidth as an incentivization scheme to increase bandwidth in the *Tor network*[15], a decentralized network for communicating anonymously and privately over the Internet.

[13] `http://slimcoin.club`
[14] `http://burstcoin.info`
[15] `https://www.torproject.org`

4. Economics in Consensus Systems

> Generally, economic agents are free to agree on the means of payment to be used to settle a transaction. However, the acceptance of money will depend on the recipient's confidence that a third party will accept [it]. Hence, the value of money lies in trust.
>
> *(ECB, The Payment System, [22])*

4.1. Economic Model

We analyze a consensus system in which a set of participants $\mathbb{P} = \{a_0, a_1, \ldots, a_{N-1}\}$ interact with each other in trying to extend a blockchain, starting from some genesis state. To this end, we propose a base model with the following specification:

- The system proceeds in rounds, where one round corresponds to a state transition, which gives rise to a time structure $\mathbb{T} = \{t_0, t_1, \ldots, t_{N-1}\}$.
- The system has an embedded monetary supply (or, alternatively, aggregated stake in the system) of M_t at time t.
- Ownership stakes in the system can be exchanged into a fiat currency at an exchange rate of X_t, thus valuing the overall supply at $V_t = M_t \cdot X_t$.
- The monetary supply grows at a rate of $s_t \in \mathbb{R}$ such that $M_{t+1} = (1 + s_t) \cdot M_t$.
- The participant to successfully propose a state transition in t is awarded some share e_t of M_t.

We propose four criteria under which the economics and design of selected systems shall be analyzed, particularly with respect to the choices and incentives of participants. We derive these criteria partially from the formal specification of a consensus protocol in section 3.3. In doing so, we establish a framework which can be used to analyze other consensus systems.

1. *Objective Consensus State*: Participants require a consistent consensus-determination function $\mathcal{D}$, which can be applied objectively to a blockchain structure to determine the system's consensus state.
2. *State Convergence*: Participants should follow the same consensus strategy $\mathcal{S}$, so as to move the consensus state forward.
3. *Communication*: Participants should be incentivized to frequently communicate among each other to share & poll state changes in the system, so as to keep everyone's consensus state current.

4. *Non-Reversibility*: It should be hard to reverse changes which lead to the current consensus state.

Objective Consensus State Participants in a consensus system require objective criteria to determine which block in a blockchain they will accept as the consensus state. The criteria used to determine the consensus state must be objective in that two independent observers reach the same conclusion about the system's state. By convention, virtually all implementations of the systems presented in section 3.3 regard the tip of the tallest[1] chain as the system's consensus state. Assuming converging state proposals, this is a natural choice because the tallest chain is the chain which is most likely to outgrow other chains. It is important to note that even though this choice appears obvious, system participants could agree on another scheme to fix the system's state.

State Convergence Our base model allows participants to propose changes to any (intermediate) state in the system. In a system with the blockchain depicted in figure 4.1, for example, change proposals to 4b, 3a, 5a (blue) are all valid. Intuitively, the consensus state should not frequently switch between different forks. As such, a consensus system must induce participants to exclusively try to extend their locally accepted consensus state. Here, we analyze the measures that systems set up to induce participants to extend their consensus state.

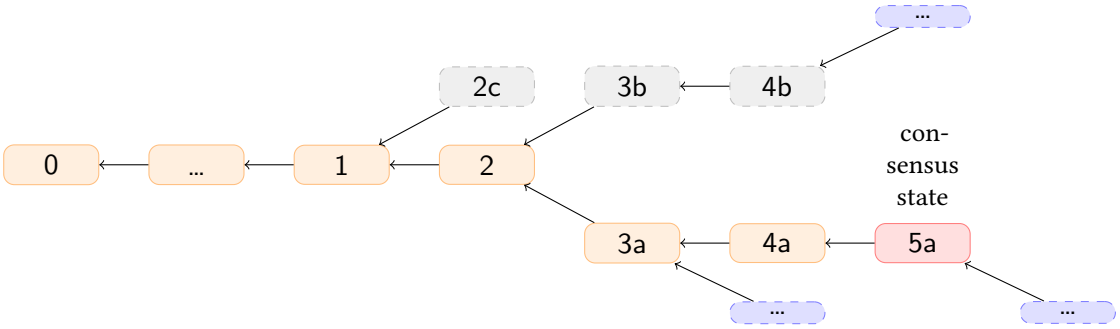


Figure 4.1.: Forked Blockchain

An important aspect in this process is the value (or, alternatively, the cost) associated with controlling a member in an economic set that yields state-transition probabilities to a participant: If an economic member can be acquired at a negligible cost, a participant could cheaply gain control of the economic set and propose arbitrary changes to any state.
We thus extend the base model to disallow such systems, and incorporate cost:

- A participant a_0 can propose changes to a *single* state with probability p_0 at cost c_0, denominated in the fiat currency. More specifically, participant a_0's success probability $P_t(a_0) = p_{(0,t)}$ is equal to her spending $c_{(0,t)}$ relative to the total spending at that time $C_t = \sum_i c_{(i,t)}$ in the network, $p_{(0,t)} = \frac{c_{(0,t)}}{C_t}$. We call the choice of which state

[1] More exactly, the tip of the chain that can be shown to have been the most *difficult* to produce is ordinarily accepted as the tallest chain.

a participant extends as her *strategy*. Additionally, we call any subset $\mathbb{P}_s \in \mathbb{P}$ of the economic set $\mathbb{P}$ for which $P_t(\mathbb{P}_s) > 50\%$ as *dominating*.

- Every participant who manages to propose a change that contributes to the consensus state receives a positive return $e_t \geq s_t > 0$ which is valid for as long as the change remains in a chain leading to the consensus state.
- Participant a_0's utility in a given round is a function of the form:

$$U_{(0,t)} = p_{(0,t)} \cdot e_t \cdot M_t \cdot X_t - c_{(0,t)} \tag{4.1}$$

In this model, each participant seeks to choose a strategy which will maximize her utility. Only if she chooses correctly, contributing to a future consensus state, can a participant hope to realize the return e_t and thus try to recover the cost incurred in proposing the change. A participant's optimal choice thus depends on the system's future consensus state which is affected by the aggregated choices of all participants in the system. As such, no participant can derive an optimal strategy without accounting for the choices of others.

We illustrate this with an example: Consider the blockchain depicted in figure 4.2 with a fork at the second-to-last change. We assume a participant a_0, with success probability $p_0 < 50\%$, in two universes. In universe a, all other participants extend state 2, while in universe b, they extend state 3.

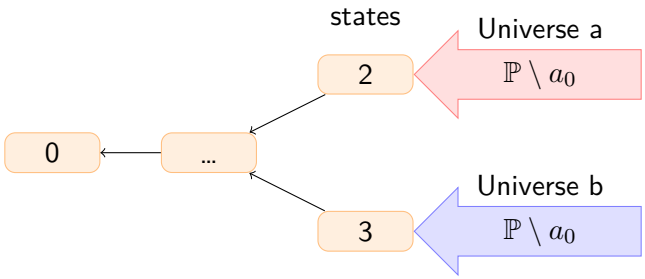


Figure 4.2.: Choice of two extension points in a forked blockchain

Participant a_0 cannot — over the long-term — succeed against a dominating set of participants. As such, a_0 must either follow the majority decision or face not contributing to a future consensus state, and thus not receiving any reward, with certainty. We highlight the expected share of the reward that a_0 and all other participants realize per round, depending on a_0's choice, in universes a and b in table 4.1.

We recognize that this resembles a coordination problem: a_0 cannot know which universe she is in without coordinating — implicitly or explicitly — with other participants. More generally, we show the expected long-term share of rewards e_t per round for a participant pursuing a different strategy than all other participants (defection), as well as following the same (honest) strategy as all other participants (cooperation) in figure 4.3. A defecting participant who herself is not a dominating set will, over the long-term, not be able to establish her changes as the consesus state. She will thus not be able to realize a share of the rewards and should instead cooperate with the majority for optimal utility. The rewards of a participant who, on the

Table 4.1.: Expected share of e_t per round

	Universe a		Universe b	
a_0 extending ...	2	3	2	3
a_o	p_1	0	0	p_1
$\mathbb{P} \setminus a_0$	$1 - p_1$	$1 - p_1$	$1 - p_1$	$1 - p_1$

other hand, is a dominating set are unaffected by her strategy choice: Any dominating set will determine the future consensus state with certainty, and can thus choose arbitrary strategies without negative long-term effects. In our analysis, we explicitly disregard potential secondary effects of a dominating set's strategy choice on X_t.

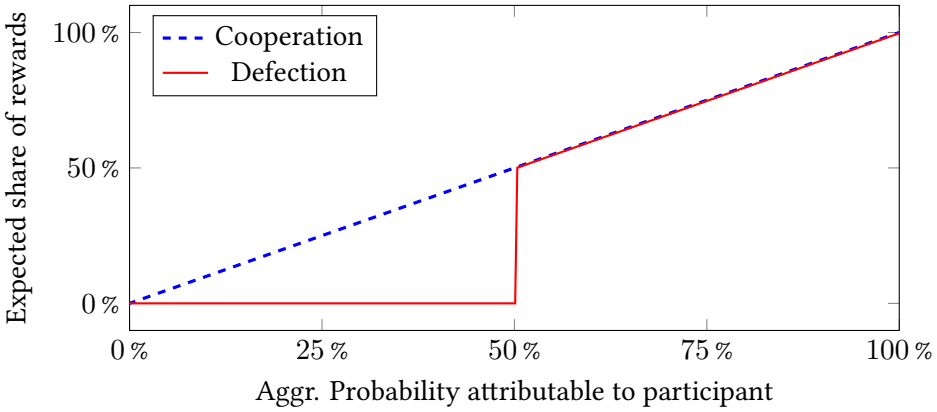


Figure 4.3.: Expected long-term share of reward earned by a single participant, either defecting or cooperating

We note that the strategy pursued by the majority of participants represents a Nash equilibrium: A single participant — whose probability is unnecessary to maintaining the majority — will always lose utility when defecting from the majority's strategy. Generalizing to the set of all participants, we recognize that the existence of a Nash equilibrium will induce all participants to pursue the majority strategy. As such, the emergence of a Nash equilibrium has self-enforcing effects, leading to a convergence of the employed strategies.

Recognizing that a strategy equilibrium can always be shifted by a dominating set of participants, we analyze the incentive compatibility of a consensus protocol (compare [6]). We call a consensus protocol *strongly incentive-compatible* if, disregarding the distribution of economic members, every participant is incentivized to follow the same consensus strategy.

Nakamoto (implicitly) argues that a consensus protocol is incentive compatible in [42]:

> "If a greedy attacker is able to [control a dominating set], he would have to choose between using it to defraud people by stealing back his payments, or using it to [gain rewards]. He ought to find it more profitable to play by the rules, [...] than to undermine the system and the validity of his own wealth."

This argument of a protocol's incentive compatibility hinges on the assumption that control of a dominating set implies future control of said set. In chapter 5.2, we highlight a bribing attack which is based on a short-term acquisition of a dominating set.

We analyze the likelihood of a defecting, non-dominating participant a_0 to "overtake" a chain of changes which is actively being extended by a dominating set, in the hope of altering the consensus state. In order for a_0 to achieve this, she must convince a sufficient number of participants from the dominating set to extend her proposed state. Figure 4.4 shows an example chain in which a participant might try to extend x_b to overtake $x_a + n$.

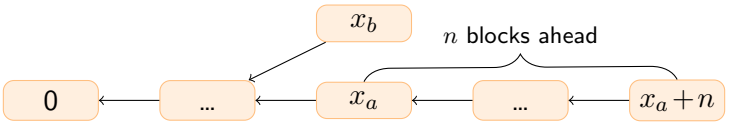


Figure 4.4.: Forked blockchain where one fork is n blocks ahead

We model the difference D_t in height of two chains as a Markov chain on $\mathbb{N}$ with transition probabilities $P(\Delta = +1) = p$ and $P(\Delta = -1) = 1 - p = q$, where Δ denotes the change of D_t per state transition of the Markov chain. In doing so, we follow [42] by applying the concepts of the *Gambler's Ruin* problem from statistics. We thus find the probability $P_{\text{Overtake}}(p, n)$ of a participant with state-transition probability p overtaking a chain that is n changes ahead and being extended by a dominating set with state-transition probability $q = 1 - p$ as (compare for example [5]):

$$P_{\text{Overtake}}(p, n) = \begin{cases} 1 & p > 0.5 \\ \left(\frac{p}{1-p}\right)^n & \text{else} \end{cases}$$

Figure 4.5 shows a_0's chances for varying starting values of n.

The key insight from this analysis is that a participant gains nothing from defecting, as his share of rewards is merely a function of his expended cost. The diminishing probability of catching up furthermore serves as a strong disincentive to any spending on chains which are not supported by the majority, even if those chains are only slightly shorter than the tallest chain in the system.

Communication From the line of reasoning established in the previous section on State Convergence immediately follows that it is in all participants' best interest to share the consensus state of their local blockchain with all other participants: As no (non-dominating) participant gains with a defecting strategy, a participant can ensure that the rest of the network is not working on extending an outdated version of the blockchain (and potentially reversing her consensus state, thus making her attempts to publish state-transitions futile). Every participant who is working on an outdated version could instead contribute to the most current consensus state, ensuring that it remains the consensus state over the long-term. For this reason, all participants are directly incentivized to frequently share their current blockchain state and poll other participants for their respective states. Through this mechanism, the network naturally maintains a high level of communication.

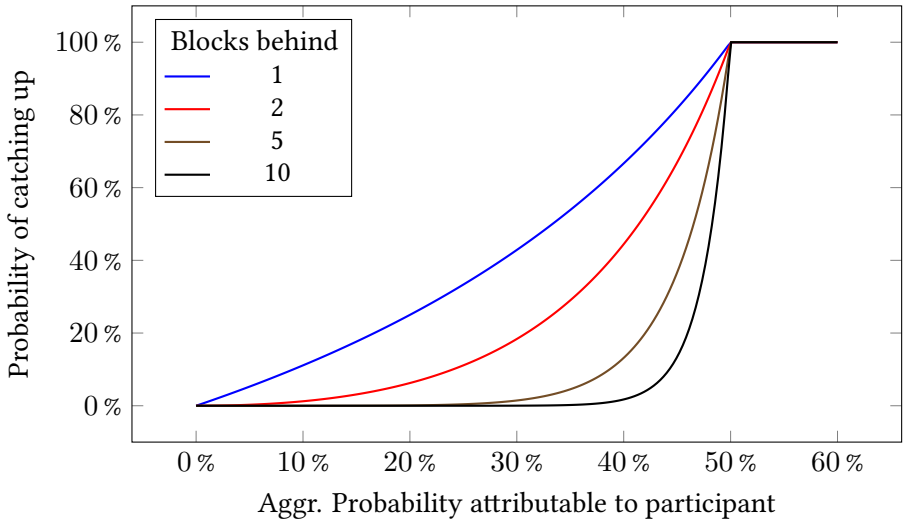


Figure 4.5.: Probability of a participant overtaking the consensus state

Non-Reversibility In the previous sections, we have established that a dominating set of participants can move the Nash equilibrium, and thus the consensus state, in the system. Since a shift in the consensus state reverses changes which had already been applied, the non-reversibility criterion reduces to the question of knowing whether any dominating set might be able to coordinate and shift the consensus state. As such, we now turn to assess how difficult it would be for a coordinated dominating set to form.

In order for a set of participants to become a dominating set, their aggregated spending must exceed that of all other participants. To understand the dynamics of this process, we analyze the incentives of an individual participant. Rearranging equation 4.1, we get:

$$\begin{aligned} U_{(0,t)} &= p_0 \cdot e_t \cdot M_t \cdot X_t - c_{(0,t)} \geq 0 \\ c_{(0,t)} &\leq \frac{c_{(0,t)}}{C_t} \cdot e_t \cdot M_t \cdot X_t \\ X_t &\geq \frac{C_t}{e_t \cdot M_t} \end{aligned} \tag{4.2}$$

Assuming e_t and M_t to be static, exogenous factors, we recognize that in the equilibrium, the total spending C_t is directly proportional to the exchange rate X_t. It follows that the spending which is required to form a dominating set, and thus the required cost to move the consensus state and reverse changes, grows linearly with X_t.

As a final observation, we note that so far we have not yet discussed which factors might drive the exchange rate X_t. Among many other potential factors which can be examined, we highlight the dependence of the exchange rate on the total spending by participants. Arguably, a stake in the monetary supply M_t is worthless if the transaction history, and hence the ownership over the stake itself, can be arbitrarily altered by anyone. This is the case if the total spending in the system is close to zero. We argue, on the other hand, that any additional spending in the

system will have a positive effect on the exchange rate, since — other aspects being equal — ownership states are harder to alter. This reduces risk in the system and thus makes ownership of an asset in the system more valuable. We illustrate the relationship between the exchange rate and the total spending in the system in figure 4.6[2].

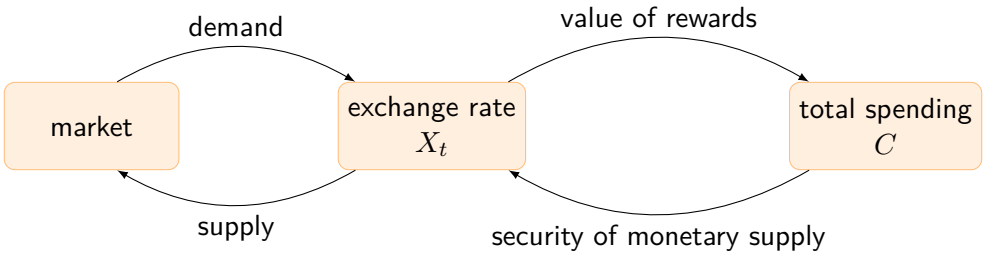


Figure 4.6.: Interdependence between total spending and exchange rate

4.2. Describing Consensus Systems

Having outlined the economic model which helps understand the incentives in consensus systems, we show how it can be related to existing consensus methods. Importantly, only consensus methods which rely on incentivization of participants can be analyzed in this model. We thus term those consensus methods *economic consensus methods* and analyze how the economic model relates to those methods. Except for Byzantine Agreement, all consensus methods reviewed in section 3.3 are of an economic nature. The main difference in these methods is the way in which they set-up the cost component $c_{(i,t)}$. From equation 4.2, we infer that the equilibrium-cost C_t^* borne by participants is equal to the prospective rewards[3] $e_t \cdot M_t \cdot X_t$.

Proof of Work The cost borne by participants in a proof-of-work system are a product of fixed cost (for acquiring microprocessors to find proofs of work) and electricity cost (for operating the hardware). The electricity required for finding a proof of work strongly depends on the microprocessor architecture. A lower bound for the amount of energy required to perform a number of calculations can be calculated using Landauer's principle [37]. This fact

[2] We provide a real-world example to establish an intuition of this relationship: A physical stock certificate might be worth less than the underlying stock itself. A reason for this may lie in required storage cost to ensure that the certificate is safely stored. This leads to a discount in the certificate's value. We thus recognize that the certificate's value depends on its security. Storage cost might involve regular payments to a custodian, whose security guarantees increase with the amount that is paid for his custody services. If the custodian were to be paid in partial ownership rights to the certificate, the security of the stock certificate would depend on the value of the partial ownership rights and, thus, its value. We also established that the value of the stock certificate depends on its security. We therefore recognize that the value of the example stock certificate and its security are interdependent.

[3] This is an assumption for an idealized setting. Importantly, expectation of future prices and various other factors can also affect the equilibrium-cost.

is unique to proof-of-work systems in that it links that system's state to an objective external resource — expended energy — which cannot be forged.

Proof of Stake A proof-of-stake system does not feature direct cost, since the economic member (system stake) is inherent to the system. As such, a proof-of-stake appears to be prone to abuse, since a manipulation of the system carries no apparent cost. Even worse, since the system relies on no exogenous factors for its economic set, economic members could potentially be used to extend multiple chains in what is known as a "nothing-at-stake" attack. We discuss these two points separately.

While a manipulation of a proof-of-stake system carries no direct external cost, an attacker does need to own a stake in the system. We thus recognize the cost incurred for participating in proof-of-stake as an opportunity cost on the stake's value: If the exchange rate X_t is low, so is the value of the stake that is bound in the system, and vice versa. We note that this resembles the exact same linear relationship between total cost and exchange rate which we derived from equation 4.2.

The "nothing-at-stake" attack — the only remaining point where proof-of-stake systems are in violation to model assumptions — is, in fact, a fundamental challenge of proof-of-stake systems: Actively used proof-of-stake systems have seen significant modifications in order to discourage nothing-at-stake attacks [11, 45].

Nothing-at-Stake Attack Since *proof-of-stake* systems do not require a meaningful expenditure of external resources, it is economically feasible to try to extend the blockchain at different blocks. Consider, for example, a participant in a proof-of-stake system which features a forked blockchain (see figure 4.7).

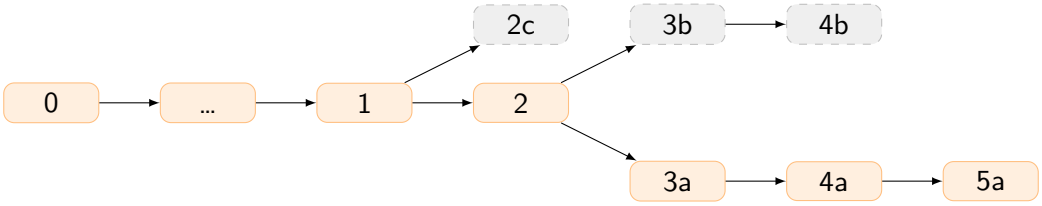


Figure 4.7.: Forked Blockchain

Ideally, all participants in the system would work on extending the blockchain at one single block. Proof-of-work systems manage this convergence of computational power through the cost involved in solving the proof-of-work puzzle. In face of the associated cost of working, all rational participants choose to spend the cost on solving the block that promises the highest reward and thus converge on one chain. In proof-of-stake, however, participants lose nothing in trying to extend the blockchain at both block 5a, the tip of the longest chain, as well as 2c and 4b. The incentives in such a system could see chains growing in parallel, thereby introducing uncertainty about the state of the overall system. Long forks of the blockchain furthermore increase the severity of double-spending attacks in the system, as it is no longer ensured that one chain will outgrow all others. Proposed mitigants against these attacks include mandated security deposits, or alternatively, temporarily locked up rewards for successfully publishing a

block, which are lost if an account is found to be working on more than one fork. Preliminary game-theoretic analyses suggest that such mitigants can properly incentivize participants to only extend one chain [10].

5. Properties & Limitations of Distributed Consensus Systems

5.1. Properties of Distributed Consensus Systems

To formalize our analysis of the presented distributed consensus systems, we summarize their most important properties. We consider the three central properties from the study of distributed systems of safety, liveness, and fault-tolerance (compare for example [15, 23]). As an additional property, we explore the notion of usefulness [48].

Our model of a distributed system consists of a set of nodes which execute a distributed state machine. The term "distributed consensus system" is thus very flexible and can refer to an arbitrary subset of nodes which communicate with each other.

Nodes in the system can experience *failure* by failing to respond to messages, diverging from the rest of the system, or behaving arbitrarily, with malicious intent. Nodes pass messages between each other to agree on a command to be executed. The set of executed commands is strictly ordered, with each executed command c having a uniquely assigned slot i. A command c can be executed on slot i when all previous commands (for slots $0 \dots i$) have been executed (such that the system's output only depends on the application of c_i to the current state) and it is clear that the system consensus will converge on c_i.

Safety A distributed consensus system enjoys safety if no two nodes will ever execute different commands c_i [39]. The safety property can be decomposed into

- *Agreement*: All nodes in the system eventually converge on the same command.
- *Validity*: The value that the system converged upon was proposed by at least one node. This validity requirement invalidates the trivial solution of all nodes always agreeing on the same one command, e.g. "do nothing".

Liveness A distributed consensus system enjoys liveness if it is able to agree on new commands without relying on any ill-behaved nodes [39]. Nodes in such a system should be free from any *stuck states*, a situation in which they are unable to make any progress, comparable to a deadlock in concurrent programming.

Fault-Tolerance A distributed consensus system enjoys fault-tolerance if the safety and liveness of smaller subsets of the system are robust to arbitrary failures of nodes in the system.

Usefulness The usefulness of a distributed consensus system is anything which affects the applicability of the system to a given task (compare [48]). As such, it is highly dependent on the intended use-case and not easily defined. We choose the following four properties as determinants of a system's usefulness:

- *Economics*: The economic implications of participating in the system. Here we assess the cost of participation in the consensus system. This is relevant because participants need to be compensated for these cost.
- *Scalability*: Here, we consider the scalability in two terms: (1) The number of nodes which can collaborate in the system. (2) The number of state-changes which can be processed per unit of time.
- *Flexible Trust*: The degree of freedom each individual node has in choosing which other nodes it trusts.
- *Simplicity*: The complexity of the system, as measured by the complexity of nodes' decisions in order to safely determine the consensus state.

Impossibility of perfectly available and safe consensus Problems of distributed consensus have been subject of extensive academic study since the 1980s [15, 35, 36, 43]. A central result from the study of distributed systems is the Fischer-Lynch-Paterson Impossibility Result [23] which states that

Theorem 1 (Fischer-Lynch-Paterson Impossibility Result) *No deterministic consensus protocol can guarantee all three of agreement, validity, and liveness in an asynchronous system.*

Any consensus systems must therefore make trade-offs in its design and choose which of the three properties can be violated.

5.1.1. Proof of Work

As outlined in section 3.3.1, it is entirely possible for two nodes in proof-of-work consensus to execute different commands. This is due to the fact that such systems have no way of ensuring synchrony between participants. Participants can therefore independently find proofs of work which they will try to extend and share with the network, thus creating a fork. We have shown in section 4.1 that — due to the embedded incentive structure — only one fork can be expected to grow beyond a certain height. We relax the definition of safety by considering executed commands as *pending* until some number of additional commands have been executed. By then, we call a command either *confirmed* or *discarded*, depending on whether or not it supported the subsequently executed commands. Applying this definition, we classify proof-of-work systems as being *probabilistically safe*. This is in line with the results from section 4.1 where we showed that a proof of work system will converge on a consensus state with increasing probability in time.

Distributed systems cannot assume synchronous communication, since nodes lack the means to synchronize through a global timestamp (see definition in section 2.5). As such, nodes in a

proof-of-work systems have to operate independently from one another in finding proofs of work. The overall system is thus free from stuck states because even one functioning node can move the system state forward. The overall network thus enjoys liveness at all times.

Summarizing section 4.1, we find that proof-of-work systems are *conditionally fault-tolerant*. Specifically, the system's fault-tolerance is conditioned on the absence of any dominating set of failed nodes.

Considering that proof-of-work is only as secure as the total spending in the system, there is a direct trade-off between a system's security and its cost. In general, this property makes proof-of-work systems relatively expensive to maintain. To illustrate this, we calculate approximated transaction cost for Bitcoin from 2014 to the end of May 2016. To this end, we analyze time series data on the monetary value of all Bitcoin transactions per day and relate them to daily rewards. The resulting data is filtered with a 30-day rolling mean to remove noise. In the assessed time period, transaction cost (in the form of mining rewards) have ranged from 5.2 to 0.8% (see figure 5.1).

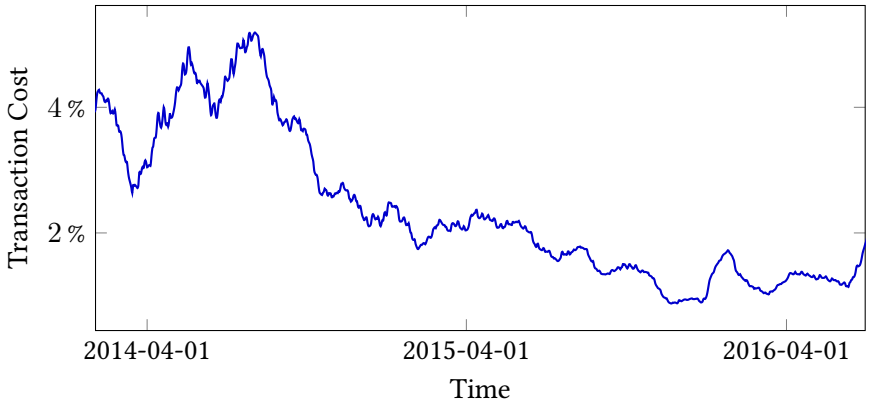


Figure 5.1.: Operating cost of Bitcoin relative to transferred monetary value

Proof-of-work systems are successfully being executed among thousands of nodes. In such a massively distributed setting, transaction throughput is greatly limited. As the number of nodes increases, so does the required time for a command to reach most nodes in the network. Empirical analyses of the Bitcoin network estimate the mean distribution time of a given block to be roughly 12 seconds [19]. Between the time of a block's discovery and its arrival at other nodes, those nodes may find a different valid block, creating conflicting blocks which lead to forks in the system (see figure 5.2). The likelihood of such forks can be expected to be proportional to the mean block distribution time divided by the mean time to block discovery. Since — at any level — only one block will end up on the long-term consensus chain, work which is put into any other blocks is wasted. This is particularly undesirable for proof-of-work systems where each published block represents a significant amount of costly work.

In light of the considerations above, Bitcoin and comparable proof-of-work systems need to balance the size of a published block (which will increase the block's distribution time and thus the chance of conflicts) and the publication frequency of a block (which will decrease the mean time to block discovery and, thus, the chance of conflicts) to maximize transactional throughput.

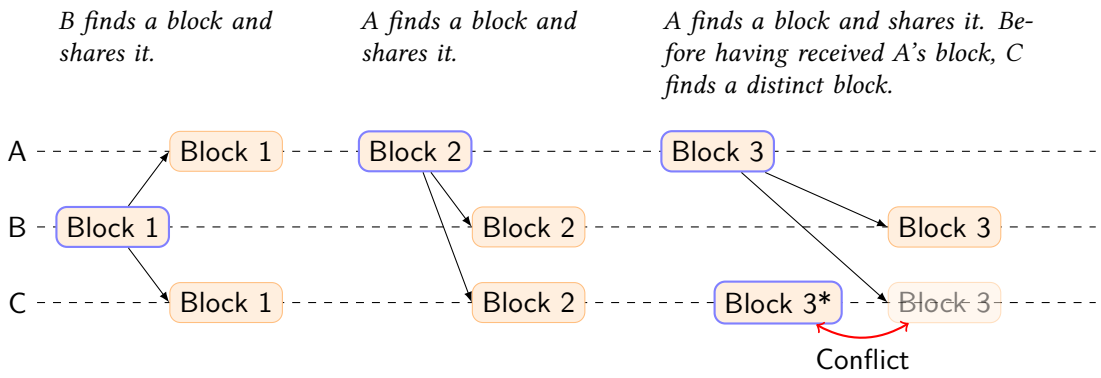


Figure 5.2.: Transmission of blocks in a distributed system. Each dotted line represents a client's blocks across time. A blue block denotes a newly found block that is being distributed. Block 3* is found shortly after Block 3, causing a conflict. Only one of these two blocks will end up on the consensus chain.

Recent proposals aim to increase the transactional throughput by enabling higher block publication frequencies. In order to offset the negative effects of abandoned blocks (see figure 5.2), they may be referenced by subsequent blocks on the long-term consensus chain [50]. The consensus chain would thus be defined as the biggest (in terms of accumulated difficulty) subtree in the blockchain, giving this mechanism its name: "Greedy Heaviest Observed Sub-Tree" (GHOST). The publishers of abandoned blocks could thus still be rewarded for their work in subsequent blocks (even if it did not affect the consensus state) and, importantly, still enforce the consensus state with their work.

In summary, we find that consensus systems must balance the consistency of the system state with the frequency with which the state is being changed. Additionally, the high publication cost of proof-of-work consensus mandates relatively low publication frequencies [55]. Consequently, in reviewing a number of proof-of-work systems (see appendix B for a list of the reviewed consensus systems), we find typical publication frequencies between 2.5 min (Litecoin) and 10 min (Bitcoin). Implementations of GHOST have been theorized to be able to enable publication frequencies close to the mean network delay (8–12 seconds) [13], enabling efficiency gains of approximately 15–60 $\times$.

A proof of work system does not afford individual nodes freedom in choosing how they determine consensus. Rather, the consensus emerges system-wide through the coordinated choices of all nodes. Hence, proof-of-work systems do not offer flexible trust.

Participation in a proof-of-work systems is relatively trivial, as a node can employ objective criteria to determine the consensus state and then only ever choose to extend this state. Assuming that participation in the network is sufficiently large, this choice provides strong security. We thus classify proof-of-work systems as having a simple design that does not require sophisticated choices on the user's part.

5.1.2. Proof of Stake

As shown in section 4.1, the fundamental design of proof-of-stake systems with regard to its incentive structure and consensus mechanism share close similarities to those of proof-of-work systems. As such, we find that proof-of-stake and proof-of-work systems largely share the safety, liveness and fault-tolerance properties outlined above.

Having established that proof-of-stake consensus can be reached much more cheaply than proof-of-work consensus (since the usage of excessive amounts of electricity is not required to secure the system), we can expect transaction cost to be equivalently lower. This is due to the fact that participants in the system do not need to be compensated for these cost and can be expected to operate the system much more cheaply. While no proof-of-stake system has yet seen wide-spread adoption, preliminary analyses suggest that proof-of-stake can be operated at two to three orders of magnitudes less cost than proof-of-work[1].

In general, we find proof-of-stake systems to offer greater scalability opportunities than comparable proof-of-work systems. We consider two important reasons for this: Firstly, proof-of-stake verification is much cheaper than proof-of-work. As such, the cost of having produced an "orphaned" block in the system is much lower. The frequency of state-transitions can thus be increased much more aggressively. Secondly, proof-of-stake systems allow for *sharding*, a technique by which the system is partitioned into a number of subsystems (compare, for example, [8]). These subsystems are then operated separately and reconciled against each other when required, yielding "horizontal scalability".

Just as with proof-of-work consensus, standard proof-of-stake consensus does not enable individual nodes to choose who to trust: As per protocol definition, the trust in the system is distributed in proportion to the stake. Proof-of-stake consensus thus offers an inflexible trust model.

The concept by which proof-of-stake consensus is determined very closely mirrors that of proof-of-work. One important difference to proof-of-work consensus is the lack of objectivity in the consensus-determination function which we highlighted in section 4.1. For this reason, any newly-joining participant in a proof-of-stake system will always have to ensure (through means outside of the system) that she is working on a valid chain, since participants could present her with (undetectably!) forged chains. In our opinion, the lack of objectivity in proof-of-stake consensus constitutes its most severe challenge.

Without any objective criteria that determine the system state, a newly joining participant must actively decide on who to trust to provide her with a valid blockchain. As such, the design of proof-of-stake systems requires participants to actively reason about trusted parties in the system and the validity of a subjective state in the system, making participation in proof-of-stake consensus slightly more complex.

Delegated Proof of Stake Delegated proof-of-stake systems concentrate the trust of the system on a smaller set of delegates. It thus differs from standard proof-of-stake consensus in that it offers flexible trust.

[1] Compare, for example: Nxt Network Energy and Cost Efficiency Analysis `nxtcommunity.org/nxt/nxt/nxt-network-energy-and-cost-efficiency-analysis` (retrieved: March 22, 2017)

By mandating delegates to bond a sufficiently large amount of system stake for a specified amount of time, some of the challenges with proof-of-stake's lack of objectivity can be remediated: Assuming a participant has access to a verified past block of sufficient recency, she can objectively come to establish system state by verifying that her past block does not conflict with the state that she has been presented with. She can do so, because she knows that delegates with bonded collateral will not forge alternative blockchains, as this would be equivalent to them forfeiting their collateral. The required recency of the block depends on the amount of time that posted collateral is bonded in the system and generally may not exceed this duration.

Under these specific conditions, delegated proof-of-stake consensus can enable an objective consensus determination, forming a consensus which has been said to be "weakly-subjective" [11].

5.1.3. Byzantine Agreement

Consensus Systems based on the concept of Byzantine Agreement (BA) feature very different trade-offs. In general, BA-systems can be fine-tuned to ensure optimal safety, or liveness, but not both. We focus our discussion on the two most important Byzantine Agreement Systems with embedded monetary supplies, *Ripple* and *Stellar*, which we presented in section 3.3.3.

Under Ripple's model of probabilistic voting, the overall system will not come to a halt, as validators will always retry to make progress. A formal protocol analysis has furthermore shown that an insufficient overlap between nodes' UNLs can cause a system to fork [1]. Ripple thus emphasizes liveness over safety.

The Stellar Consensus Protocol precisely defines the conditions which a node's quorum slices must fulfill for it to be considered to be part of the system. Stellar's model of federated voting in conjunction with the specification of quorum slices furthermore ensures that the system cannot fork. We can thus judge Stellar to be safe. This comes at the cost of system liveness: A network operating under the Stellar Consensus Protocol can come to a halt in a system-wide stuck state [39].

Consequently, we find that the Stellar and Ripple protocol solve similar problems with completely different safety-liveness trade-offs. These choices are likely to affect the protocols' long-term applicability in different process, as a stale or stuck system are likely to have vastly different associated cost.

Similar to proof-of-stake consensus, Byzantine agreement is almost costless to participating nodes[2].

An important difference to proof-of-work and proof-of-stake consensus protocols lies in the scalability properties of Byzantine agreement: Byzantine agreement systems have been successfully deployed in operating a distributed ledger, achieving transaction throughput of 4,500 transactions/s (two – three orders of magnitude higher) with 64 nodes. Adding more nodes, however, quickly degrades network performance, as all nodes need to maintain a higher level of communication[3] [18]. This finding is in line with the low number of nodes that currently operate Ripple and Stellar[4].

[2]Empirical analyses estimate transaction cost of approximately $4 \cdot 10^{-7}$\$ at full throughput.

[3]For example, PBFT a prominent byzantine consensus system has a quadratic complexity of message passing in the number of participating nodes [15].

[4]Lists of validators (as of March 22, 2017). Ripple: `https://validators.ripple.com/#/validators`; Stellar:

Byzantine agreement protocols allow nodes to freely choose which other nodes they want to trust. This is, for example, realized through the UNL (Ripple) or quorum slices (Stellar). Nodes are thus afforded flexible trust. Importantly, however, these trust decisions are not only flexible, but also necessary in order for the node to be able to correctly participate in the system. If a node, for example, chooses to only rely on a statement if a non-functioning, stuck node supports it, it will equivalently be stuck. The flexible trust property of Byzantine agreement systems thus requires nodes to reason about which other participants in the system might be trustworthy. In the light of trade-offs between the size of the overlap of individual nodes trust sets (see section 3.3.3), we classify participation in a Byzantine agreement system as being complex.

5.2. Failure Modes of Distributed Consensus

Having reviewed the functionality and properties of various distributed consensus systems, we now examine ways in which distributed consensus in such systems can fail or be exploited. To this end, we describe attack methods which can be used to subvert the distributed consensus in some way. Specifically, we introduce three classes of attacks which summarize ways in which consensus systems can be subverted:

- *Attacks on Consensus State*: An attacker objectively changes the consensus state in the system.
- *Attacks on Consensus Strategy*: An attacker induces other participants to change their consensus strategy.
- *Attacks on Network Topology*: An attacker distorts the natural topology of the network of participating nodes, for example by removing or adding nodes.

Attacks on Consensus State

51% Attack As analyzed in section 3.3.1, any attacker who controls more than 50% of the overall state-transition probabilities can arbitrarily change the system's state. Without any other measures, the security against 51% attacks is thus the cost of acquiring the economic set required to gain dominance. Ideally, this cost should be prohibitively expensive so as to make this attack practically impossible. Critically, an attacker is unlikely to launch a "self-financing" attack strategy, but must instead suffer economic cost for launching an attack. This is due to two main reasons:

- *Detection time*: A 51% attack is likely to be noticed quickly[5] (in a matter of hours [30]). It is thus improbable that an attacker would be able to sell a sufficient stake in the system before the attack is noticed. As soon as the attack is noticed, the exchange rate will deteriorate rapidly, making any remaining stake worthless.

`http://104.196.96.245/`

[5]Historically, smaller proof-of-work consensus systems have seen attacks which were detected quickly. Compare, for example: `coindesk.com/feathercoin-hit-by-massive-attack/`

- *Liquidation challenges*: A big sell-off in the time between a successful attack and its detection will already draw down the exchange rate.

A 51% attack is thus unlikely to be launched with a direct financial motive.

Long Range Attack Long-range attacks try to subvert the consensus state with a fork of the blockchain from a state of relatively short height [9, 45]. We illustrate a long-range attack from a blockchain's genesis state in figure 5.3.

In order for the attacker to succeed, he needs to find a way to very quickly generate changes which he can apply to his fork. One method which can be used exploits the fact that the attacker has full control over all blocks (f1 to f12 in figure 5.3). If the difficulty of finding a new block recursively depends on the state of the chain which is extended, the attacker can manipulate this state to enable him to quickly find new blocks. Once the attacker has grown a fork of sufficient height, he can share it with other participants in a system. If the participants have no way of distinguishing the "honest" consensus chain from the maliciously created fork, they will accept the longer fork. An attacker can thus reverse previous transactions and abuse the system.

In this context, we highlight two system designs which are susceptible to a long-range attack [9]:

- *Algorithmic Proof-of-Work*: A modification to standard proof-of-work presented in section 3.3.1. Instead of partial preimage searches, participants are required to execute algorithms which are stored in prior blocks. To exploit this system, an attacker adds specially-prepared algorithms to newly-generated blocks. These algorithms are computationally complex, but can be quickly executed with knowledge about terminating conditions ("short-cuts") of the algorithm in question. As he progressively adds blocks to his private fork, the attacker gets to execute algorithms which were added by himself to prior blocks. If he knows these algorithms' terminating conditions, he can progressively increase the required time to find a new block. An outsider has no way of determining whether the algorithms which were added by a third-party are specially-prepared ones or regular ones. As such, it is impossible to determine whether or not a given chain is the result of a long-range attack.

- *Naïve Proof-of-Stake*: An attacker builds a private fork in which he progressively transfers the stakes from other participants in the system to himself. Since his likelihood of finding a block is proportional to his stake in the system, he can thus "gain speed" as he accumulates more wealth.

Short Range Attack A short-range attack is an attempt to change the consensus state with a focus on reversing changes that were added after a specific block. Short-range attacks thus focus on forking the blockchain at a similar height than that of the accepted consensus block (figure 5.3). We derive the probability of a participant on a shorter chain catching up with a consensus state in section 4.1. Extending these results, we derive the probability of a change being reverted by a short-range attack after n additional blocks have been applied. We assume

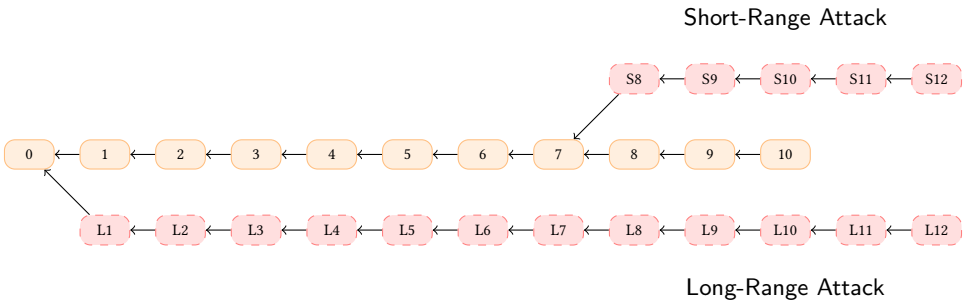


Figure 5.3.: Short- and Long-Range Attack on a Blockchain

an attacker with aggregated state-transition probabilities of $q = 1 - p$. As soon as the respective change has been applied, both the attacker and the remaining participants work on extending their respective chains. The difference in height between these two chains — after n blocks have been added — follows a Poisson distribution with mean $n\frac{q}{p}$ [42]. The likelihood of success of a short-range attack is a function of the number of blocks which have been added and the attacker's aggregated state-transition probabilities (see figure 5.4).

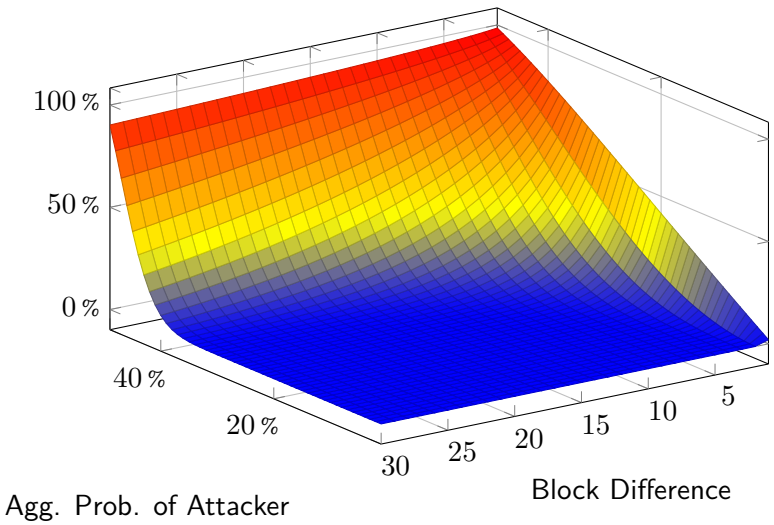


Figure 5.4.: Chances of an attacker on a shorter chain overtaking the consensus chain

Attacks on Consensus Strategy

P+Epsilon-Attack We present a method for an attacker to manipulate consensus systems which rely on coordination between participants [12]. To this end, we review a game in which participants vote on one element from a set of choices. Without loss of generality, we consider a game with two choices a and b. If a participant votes on the element with the most votes, she is awarded a reward P. Otherwise, she receives nothing. Table 5.1a outlines the game's

payoff from the view of a single participant. This game is akin to coordination games which we reviewed in section 4.1.

Table 5.1.: Voting Game: Payoff depends on voting behavior

(a) Standard voting game

		Participant a	Participant b
Others	a	P	0
	b	0	P

(b) Modified voting game: $P + \epsilon$ attack

		Participant a	Participant b
Others	a	P	$P + \epsilon$
	b	0	P

We now imagine an attacker who tries to manipulate the game's outcome. The attacker promises participants who vote b a payment of $P + \epsilon$ if — and only if — the majority voted a. If we overlay this promise with the game's original payout profile, a Nash equilibrium emerges in (b, b) (see table 5.1b): The strategy "vote b" weakly dominates "vote a" for all participants. For this reason, everyone votes b and realizes a return of P. The attacker has thus manipulated the game's outcome without incurring any cost.

Bribe Attack As another form of controlling the consensus strategy, an attacker might offer to "rent" participants' state transition probabilities [6]. Here, renting out state transition probabilities refers to a participant temporarily yielding control over her economic set to another party. Controlling this economic set, the party is then able to reap rewards from publishing blocks and may follow any consensus strategy.

In order to convince any participant to rent out her state transition probabilities, the attacker must promise a greater reward than what can be expected from following the protocol. An attacker would thus lose — without any further steps — on expectation from merely following the standard consensus protocol. A bribe attack is thus only economically attractive if the attacker manages to utilize his economic set to subvert the consensus state. For example, he might seek to revert a past payment of his in a short-range attack.

In summary, we find that a bribe attack can give an attacker short-term control over a dominating economic set. Since the attacker can hold onto the economic set for only short periods, he is able to construct a situation where he disassociates the long-term incentive of ensuring the system's stability and the state-transition probabilities associated with economic members.

Attacks on Network Topology

Sybil Attack Sybil attacks are a method of manipulating distributed systems by means of ill-behaved nodes which are added to the system [20]. A Sybil attack is thus always an exploitation of the trust topology of the network: If a subset of a distributed consensus system is not fault-tolerant (because it trusts ill-behaved participants), it can theoretically be manipulated in a Sybil attack. Various distributed systems incorporate heuristics to ensure an adequate topology of

the network. Bitcoin, for example, by default only connects to one IP-address per /16-subnet[6], thus requiring an attacker to place ill-behaved nodes in many different places in the network.

5.3. Susceptibility of Consensus Methods to Attacks

Attacks on Consensus State

51% Attack The cost of performing a 51% attack is controlled by the cost of acquiring a majority of the economic set. We can thus generally expect proof-of-stake systems to be much more robust against such attacks than proof-of-work systems: The cost of a 51% attack in proof-of-work systems is proportional to the distributed rewards over the amount of time required to acquire a majority of computing power in the system. For proof-of-stake systems, the cost of a majority in the economic set is equal to the value of a majority of the stake in the system. Assuming yearly proof-of-work rewards of 4%[7], we can thus estimate the cost of a 51% attack on a proof-of-stake system, carried out over — conservatively — one year, to be twenty times as high as an attack on a comparable proof-of-work system.

Long-Range Attack Standard proof-of-work consensus is very much robust to long-range attacks. This is due to the fact that the computational cost of performing a long-range attack is equal to the overall computational cost expended in building a given chain (see chapter 4). Since proof-of-stake systems lack an objective consensus-determination function, they are highly susceptible to long-range attacks. More concretely, due to the fundamental nothing-at-stake challenge, an attacker can costlessly "simulate" many different blockchain histories in a proof-of-stake system [45].

Short-Range Attack The fundamental cost of a 51% attack also applies to short-range attacks. Proof-of-stake systems can be secured against "costless simulation" attacks over short-ranges by collateralizing stake in the system (see section 3.3.2). This idea is based on the fact that one can trust participants with collateralized stakes in the system for exactly the amount of time that their collateral is locked up in the system. If the duration of collateralization, for example, is one month, any block which is believed to have been published in the last month can be trusted. Short-Range attacks can thus be resisted if participants in the system if participants in the system "police" collateralized stakeholders if they are found to be simulating different chains. This control of participants in the system ceases with the move of any locked up funds [45].

Attacks on Consensus Strategy

Attacks on the consensus strategy rely on an attacker's ability to manipulate the default incentives of other participants in the system. As such, it is not directly related to the consensus strategy employed in a consensus protocol. In fact, both proof-of-work, as well as proof-of-stake consensus are susceptible to p+epsilon and bribe attacks. The only difference in the susceptibility

[6] Bitcoin source code hosted on GitHub: https://github.com/bitcoin/bitcoin/blob/6caf3ee061a86d89b965dc8a61a95d1f34015805/src/net.cpp#L1636

[7] The Bitcoin inflation rate currently sits at 4.09% (July 10)

of both protocols may lie in the exact incentives of participants. We argue that proof-of-stake systems are more susceptible to attacks on the consensus strategy than comparable proof-of-work systems. This is due to the fact that their rewards for successfully published blocks are generally set lower than in comparable proof-of-work systems, as reviewed in section 5.1.2.

Attacks on Network Topology

Sybil Attack All reviewed consensus methods can be — to varying extent — manipulated in a Sybil attack. In proof-of-work consensus, added nodes can disrupt communication in the network by failing to pass on information or passing on incorrect information. As outlined in chapter 4, they will, however, not be able to change the consensus state or convince nodes of a manipulated consensus state.

Proof-of-stake consensus, lacking objectivity in its state-determination, is more susceptible to Sybil attacks. Newly added nodes can try to provide participants with maliciously created states. Since participants are unable to distinguish any two states (since they lack objective criteria), they have to decide on a state based on some criterion[8]. One example of such a criterion is participants choosing the most prevalent state in the system. Under this criterion, a Sybil attack's succeeds when more Sybil nodes than honest nodes have been added.

Lastly, Byzantine Agreement has been shown to be robust to Sybil attacks where fewer than 1/3 of all nodes are faulty within a given window of time [15]. If a Sybil attack manages to add nodes amounting to more than 50% of an existing network's size, it can therefore always succeed in compromising Byzantine Agreement.

[8] Alternatively, they could randomly choose a state, making the attack's success likelihood approximately proportional to the share of Sybil nodes in the system.

6. Discussion

6.1. Economic Impact

> Bitcoin will do to banks what email did to the postal industry
>
> *(Rick Falkvinge, Founder of the Swedish pirate party)*

As can be seen from this work, distributed consensus systems can be used to describe very generic systems. As such, they have the potential to significantly change the technical underpinnings of the current financial system. We substantiate this claim with three observations:

1. *Potential for large cost savings*: Annual revenues in areas that can directly benefit from distributed consensus technologies are estimated to be approximately 40 bn USD [7, 38]. While there is significant uncertainty around such estimates, there is clear potential for increased efficiency in financial markets. In the context of existing processual inefficiencies, these numbers serve to illustrate potential efficiency gains which can be realized through distributed consensus systems.

2. *Established financial companies allocate significant activity and attention to distributed consensus*: The number of announcements related to distributed consensus/blockchain technology more than doubled from Q3 2015 to Q1 2016[1]. R3, a fintech startup backed by a consortium of large financial institutions, recently conducted experimental tests with distributed consensus technology across forty banks[2]. The activity by established companies serves to show that the potential of distributed technology has found recognition in the financial industry.

3. *Growing investments into distributed consensus technology*: In a recent report, Greenwhich Associates estimate investments into distributed ledger technologies to have exceeded 1 bn USD[3], with additional growth expected. Venture capital funding of startups in the distributed consensus/blockchain space have also surpassed 1 bn USD[4], with the influential venture capital firm *Andressen Horowitz* including blockchain technology as one of 16 important trends in technology[5].

[1] `http://coindesk.com/state-of-blockchain-q1-2016/`
[2] `http://coindesk.com/r3-consortium-banks-blockchain-solutions/`
[3] `http://greenwich.com/fixed-income-fx-cmds/blockchain-adoption-capital-markets`
[4] `http://www.coindesk.com/bitcoin-venture-capital/`
[5] `http://a16z.com/2015/01/22/bitcoin-and-blockchain/`

6.2. Impact of Quantum Computing

> Imagine that it's fifteen years from now. Somebody announces that he's built a large quantum computer. RSA is dead. DSA is dead. Elliptic curves, hyperelliptic curves, class groups, whatever, dead, dead, dead. So users are going to run around screaming and say "Oh my God, what do we do?"
>
> *(Introduction to Post-Quantum Cryptography, `http://pqcrypto.org`)*

Quantum Computing refers to a field in Computer Science which studies how a computer can utilize phenomena of quantum mechanics, such as quantum entanglement. Theoretical work predicts that a large-scale quantum computer will be able to solve certain classes of problems much more efficiently than traditional computers. Importantly, the computational difficulty of these problem classes lies at the heart of many modern cryptograhic methods, some of which were reviewed in chapter 2. As such, the existence of a quantum computer would mandate changes to current cryptographic security procedures.

To this date, there has been no public presentation of a sufficiently sized quantum computer that is able to bring about practical benefits. Nevertheless, quantum computing remains a field of active research and investment[6]. In the following, we briefly examine how the existence of a quantum computer impacts distributed consensus systems and the cryptography on which they are based.

Public-Key Cryptography: Shor's algorithm Shor's algorithm is an important theoretical result from the study of quantum computers [49]. It is a generalized algorithm for performing prime factorization on integers on a quantum computer in polynomial time, rather than exponential time[7] required by the most efficient generalized algorithm today. Using Shor's algorithm on a sufficiently large quantum computer, it would be possible to break RSA and similar methods of asymmetric cryptography. This would allow an attacker to control the funds on most blockchains (since the attacker would be able to forge arbitrary signatures from public keys).

Preimage Search: Grover's algorithm Using Grover's algorithm, quantum computers could realize a quadratic efficiency-gain for the kind of preimage searches employed in both proof-of-work and proof-of-stake[8] [26, 27]. Such an efficiency gain is equivalent to a halving of the difficulty parameter d in use in proof-of-work or proof-of-stake consensus systems. This is equivalent to an attacker having twice his original computing power. Considering that Grover's

[6] Recent *Nature* publication describing empirical results in scaling quantum computers [54]; Leaked information on an NSA-led research program which focused on quantum computing `https://www.washingtonpost.com/apps/g/page/world/a-description-of-the-penetrating-hard-targets-project/691/`

[7] The exact asymptotic bounds have been shown to be: $\mathcal{O}((\log N)^2 (\log\log N)(\log\log\log N))$ for Shor's algorithm [3] vs. $\mathcal{O}\left(\exp\left(1.9(\log N)^{\frac{1}{3}}(\log\log N)^{\frac{2}{3}}\right)\right)$ for the Number Field Sieve, the standard algorithm employed currently [16].

[8] Grover's algorithm is applicable to any ***NP**-complete* problem, that is, a problem that cannot be solved in polynomial time. The difficulty of such problems ordinarily grows exponentially in their input, making such problems especially hard to solve for large inputs.

algorithm thus does not allow an attacker to easily overpower an existing network, preimage attacks appear to be a smaller threat than the breaking of asymmetric cryptography described above.

Cryptography under the presence of quantum computers is a field of active research. The European Union, for example, recently funded the project "Post-quantum cryptography for long-term security"[9] with 3.8 mn EUR. The project seeks to establish cryptographic systems which remain secure under attacks from quantum computers. Methods which have been shown to withstand attacks by quantum computers could then replace existing cryptographic systems, for example as they are currently being used in consensus systems. Given the potential destructive potential of quantum computing to the security of public consensus systems (among many other areas), research in this field is of great practical importance.

6.3. The Role of Regulation

> I say this, however, with a caveat that we or rather the global regulatory community elsewhere have not taken a final stance on the use of distributed ledger technology. It is important to highlight here that [the] Financial Stability Board has already started consultations on developing [a] better understanding of the intricacies involved.
>
> *(Subhash Sheoratan Mundra, deputy governor of the Reserve Bank of India, in a speech about "Disruption, Innovation, and Competition" in the Indian Banking Sector)*

The advent of distributed consensus systems in our economy marks an important shift for the work and scope of financial regulators and central banks. We highlight two areas which pose particular challenges for regulators: Firstly, an increased need for regulators to steer the development of financial infrastructure. Secondly, a gradual loss of regulatory control over financial transactions and monetary policy as cryptocurrencies gain traction.

Regulatory involvement in development of distributed technology The current centralized model of financial infrastructure lends itself very well to regulatory oversight, since there is usually a single source of financial transactions and encoded rules. As financial infrastructure evolves and shifts to a decentralized model, this single source will be replaced by multiple distinct sources, possibly with different sets of transactions and different encoded rules. Decentralized infrastructure thus poses challenges for consistent regulatory oversight. A permissionless distributed infrastructure would furthermore exacerbate this situation as regulators would be left with no clear means of tying digital identities to real-world persons. In order to allow for effective enforcement of regulations in this new setup, regulators thus need to oversee the development of this infrastructure, so as to ensure that there continues to be a means for adequate financial regulation.

Considering regulator's need for involvement in this area, it is important to note that the potential impact of distributed consensus technology will strongly depend on the type of influence that regulators take. The regulatory framework on distributed technology, which

[9] `http://cordis.europa.eu/project/rcn/194347_en.html`

will need to be defined by legislators and financial regulators globally, will equally affect the potential effects on the financial industry. In line with the vastly different political agendas of various governments, policies governing the many aspects of distributed technology are likely to diverge substantially. Yet, in order to fully realize the benefits of distributed technology, market participants will require a consistent and global legal framework.

An interesting first example of regulatory intervention against a distributed technology firm emerged on May 5, 2015. The *Financial Crimes Enforcement Network* of the United States imposed a civil fine of 700 k USD on Ripple, the company behind the equally-named variant of Byzantine Agreement reviewed in chapter 3, for not maintaining adequate anti-money laundering processes and failing to appoint a compliance officer [33].

Loss of control over monetary policy A second aspect is the emergence of cryptocurrencies as value stores which reside outside of monetary control, but are perfectly fungible and thus well-suited (to varying degrees) for payments and international trade. In this context, cryptocurrencies have enabled two developments:

1. Cryptocurrencies are ordinarily set up as permissionless systems. They thus enable individuals to trade with an increased level of privacy and without regulatory control. Regulators ordinarily mandate banks to take measures to detect and prevent anti money-laundering and terrorist financing. Since any individual with an Internet connection can participate in a permissionless system, it is particularly appealing to criminals wishing to conceal their identity. In fact, it is trivial to create multiple different identities and to freely trade among these within the system, without a clear way of externally tying these identities together. As such, regulators will want to put increased focus on exchanges which enable trade between crypto- and fiat currencies to enforce their policies.

2. Exisintg cryptocurrencies also do not allow for adjustable inflation of its monetary base by outside parties. Rather, the inflation of most systems is governed by a deterministic function that can only be changed by controlling a dominating economic set. It follows that regulators will be unable to enforce monetary policy in such currencies (without acquiring a dominating economic set). Participants in a cryptocurrency have the power to destroy funds, thereby introducing a form of deflation in the monetary base. This is akin to the burning of money which constitutes a crime in many legislations. To illustrate this process, we burn an amount of 0.001 Bitcoin and document the process in appendix D.

National regulatory regimes have — so far — responded very differently to the growing use and importance of cryptocurrencies. This fact becomes evident when considering the stance of two countries, namely Russia and Estonia, towards Bitcoin: Russia's "Internet advisor" recently suggested that any interaction with Bitcoin constituted a crime[10]. The Monetary Authority of Singapore (MAS), on the other hand, stated that "[w]hether or not businesses accept Bitcoins in exchange for their goods and services is a commercial decision in which MAS does not intervene." [51].

[10] "Bitcoin is not the first one, there are other settlement means. Accepting bitcoin as a payment for anything is unacceptable because it is a crime." `http://coindesk.com/receiving-bitcoin-payments-is-a-crime-says-russian-presidential-advisor/`

6.4. Conclusion

In this work, we scrutinize the design of distributed consensus systems and, specifically, different methods of establishing consensus among multiple parties. After having been invigorated by Bitcoin's proof-of-work consensus, distributed consensus has become a highly dynamic field with much work being committed outside of traditional academia. Accordingly, many aspects of distributed consensus lack formal analysis and concrete criteria under which newly proposed systems can be analyzed.

We propose four criteria under which the effectiveness of distributed consensus methods can be analyzed under economic considerations. Based on these criteria, we show how the incentives of economic agents in a consensus system accomplish a stable consensus state. Additionally, we are able to use these criteria to derive challenges in proof-of-stake consensus.

In summary, we find that distributed consensus provides a technological foundation for new innovations, for example in financial markets, where they can be expected to positively affect the quality and price of financial services for the economy as a whole. In the short-term, more real-world experience will be required to uncover potential pitfalls in the implementation of distributed consensus technology in existing processes and infrastructure. Considering the relevance of the global financial economy, whose fragility could be severely impacted if a hastily implemented consensus system were to fail, this phase of testing requires special attention by regulators, as well as academic and industry specialists. We additionally discuss the impact of quantum computing on existing cryptographic methods with respect to the security of consensus systems. The development and subsequent theoretical analysis — using methods from economics and computer science — of distributed consensus systems will therefore continue to be of high practical relevance.

Bibliography

[1] Frederik Armknecht et al. "Ripple: Overview and outlook." In: *Proceedings of International Conference on Trust & Trustworthy Computing*. Vol. 9229. 2015, pp. 163–180. DOI: 10.1007/978-3-319-22846-4_10.

[2] Adam Back. *Hashcash - A Denial of Service Counter-Measure*. 2002. URL: http://hashcash.org/papers/hashcash.pdf.

[3] David Beckman et al. "Efficient Networks for Quantum Factoring." In: *Physical Review A* 54.2 (1996), pp. 1034–1063. DOI: 10.1103/PhysRevA.54.1034. arXiv: 9602016 [quant-ph].

[4] Iddo Bentov et al. "Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake." In: *Cryptology ePrint Archive* 452 (2014), pp. 1–19. DOI: 10.1145/2695533.2695545.

[5] Joseph Blitzstein and Jessica Hwang. *Introduction to probability*. 2015.

[6] Joseph Bonneau et al. "Why buy when you can rent? Bribery attacks on Bitcoin consensus." 2014.

[7] Francois Brochet and Gregory S Miller. *The Capital Markets Industry — The Times They Are A-Changin'*. Tech. rep. Oliver Wyman \& The SWIFT Institute, 2015. URL: http://www.oliverwyman.com/insights/publications/2014/sep/the-capital-markets-industry.html.

[8] Vitalik Buterin. "Ethereum: Platform Review." 2016. URL: http://r3cev.com/s/Ethereum%7B%5C_%7DPaper-97k4.pdf.

[9] Vitalik Buterin. *Long-Range Attacks: The Serious Problem With Adaptive Proof of Work*. 2014. URL: http://blog.ethereum.org/2014/05/15/long-range-attacks-the-serious-problem-with-adaptive-proof-of-work/.

[10] Vitalik Buterin. *On Stake*. 2014. URL: http://blog.ethereum.org/2014/07/05/stake/.

[11] Vitalik Buterin. *Proof of Stake: How I Learned to Love Weak Subjectivity*. 2014. URL: blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity/.

[12] Vitalik Buterin. *The P + epsilon Attack*. 2015. URL: http://blog.ethereum.org/2015/01/28/p-epsilon-attack/.

[13] Vitalik Buterin. *Toward a 12-second Block Time*. 2014. URL: https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time/.

[14] Vitalik Buterin. *Understanding Serenity, Part 2: Casper*. 2015. URL: http://blog.ethereum.org/2015/12/28/understanding-serenity-part-2-casper/.

[15] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance." In: *Proceedings of the Symposium on Operating System Design and Implementation* February (1999), pp. 1–14. DOI: 10.1145/571637.571640.

[16] Don Coppersmith. "Modifications to the Number Field Sieve." In: *Journal of Cryptology* 6.3 (1993), pp. 169–180. ISSN: 1432-1378. DOI: 10.1007/BF00198464. URL: http://dx.doi.org/10.1007/BF00198464.

[17] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design*. Vol. 4. 2012, p. 772. ISBN: 0321263545.

[18] Kyle Croman et al. "On Scaling Decentralized Blockchains." In: *3rd Workshop on Bitcoin and Blockchain Research*. International Financial Cryptography Association, 2016.

[19] Christian Decker and Roger Wattenhofer. "Information propagation in the Bitcoin network." In: *13th IEEE International Conference on Peer-to-Peer Computing, IEEE P2P 2013 - Proceedings*. 2013. ISBN: 9781479905218. DOI: 10.1109/P2P.2013.6688704.

[20] Jr Douceur. "The sybil attack." In: *Peer-to-peer Systems*. Springer Berlin Heidelberg, 2002, pp. 251–260. ISBN: 978-3-540-44179-3.

[21] Karl J O Dwyert and David Malone. "Bitcoin Mining and its Energy Footprint." In: *Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014). 25th IET* (2013), pp. 280–285.

[22] European Central Bank. *The payment system: Payments, Securities and Derivatives, and the Role of the Eurosystem*. 2010. ISBN: 978-92-899-0633-3.

[23] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. "Impossibility of distributed consensus with one faulty process." In: *Journal of the ACM* 32.2 (Apr. 1985), pp. 374–382. ISSN: 00045411. DOI: 10.1145/3149.214121. URL: http://portal.acm.org/citation.cfm?doid=3149.214121.

[24] Glenn Fowler et al. *The FNV Non-Cryptographic Hash Algorithm*. 2014. URL: http://tools.ietf.org/html/eastlake-fnv-03.

[25] Mainak Ghosh and Miles Richardson. *A TorPath to TorCoin: Proof-of-Bandwidth Altcoins for Compensating Relays*. 2014. URL: http://www.robgjansen.com/publications/torpath-hotpets2014.pdf.

[26] Lov K Grover. "A fast quantum mechanical algorithm for database search." In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96*. 1996, pp. 212–219. DOI: 10.1145/237814.237866. arXiv: 9605043 [quant-ph].

[27] Lov K Grover. "From Schrödinger's equation to the quantum search algorithm." In: *American Journal of Physics* 69.7 (2001), p. 769. DOI: 10.1119/1.1359518. arXiv: 0109116 [quant-ph].

[28] Don Johnson, Alfred Menezes, and Scott Vanstone. "The Elliptic Curve Digital Signature Algorithm (ECDSA)." In: *International Journal of Information Security* 1.1 (2001), pp. 36–63. DOI: 10.1007/s102070100002. arXiv: 1011.1669v3.

[29] Leslie Kaelbling et al. "6.01SC Introduction to Electrical Engineering and Computer Science I." In: *Massachusetts Institute of Technology: MIT OpenCourseWare*. 2011.

[30] Ghassan O Karame et al. "Misbehavior in Bitcoin: A Study of Double-Spending and Accountability." In: *ACM Transactions on Information and System Security (TISSEC)* 18.1 (2015), p. 2. DOI: 10.1145/2732196.

[31] Sunny King. *Primecoin: Cryptocurrency with Prime Number Proof-of-Work*. 2013. URL: http://primecoin.io/bin/primecoin-paper.pdf.

[32] Sunny King and Scott Nadal. *PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake*. 2012. URL: http://ppcoin.org/static/ppcoin-paper.pdf.

[33] Trevor Kiviat. "Beyond Bitcoin: Issues in Regulating Blockchain Transactions." In: *Duke Law Journal* 65.3 (2015), pp. 569–608. DOI: 10.1525/sp.2007.54.1.23..

[34] Jae Kwon. *TenderMint : Consensus without Mining*. 2014. URL: tendermint.com/docs/tendermint.pdf.

[35] Leslie Lamport. "The Part-Time Parliament." In: *ACM Transactions on Computer Sys-tems* 16.2 (1998), pp. 133–169. DOI: 10.1145/279227.279229.

[36] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem." In: *ACM Transactions on Programming Languages and Systems* 4.3 (1982), pp. 382–401. DOI: 10.1145/357172.357176.

[37] Rolf Landauer. "Irreversibility and Heat Generation in the Computing Process." In: *IBM Journal of Research and Development* 5.July (1961), pp. 183–191. DOI: 10.1147/rd.53.0183.

[38] Michael Mainelli and Alistair Milne. *The Impact and Potential of Blockchain on the Securities Transaction Lifecycle*. 2016.

[39] David Mazières. *The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus*. 2015.

[40] Alfred J. Menezes, Paul C. Van Oorschot, and Scott a. Vanstone. *Handbook of Applied Cryptography*. Vol. 106. CRC press, 1997, p. 780. ISBN: 0849385237.

[41] Andrew Miller et al. "Nonoutsourceable Scratch-Off Puzzles to Discourage Bitcoin Mining Coalitions." In: *(preprint)* (2015).

[42] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System." In: *Consulted* (2008), pp. 1–9. URL: http://bitcoin.org/bitcoin.pdf.

[43] Marshall Pease, Robert Shostak, and Leslie Lamport. "Reaching agreement in the presence of faults." In: *Journal of the ACM (JACM)* 27.2 (1980), pp. 228–234.

[44] C Percival. "Stronger key derivation via sequential memory-hard functions." In: *Self-published* (2009), pp. 1–16. URL: http://tarsnap.com/scrypt/scrypt.pdf.

[45] Andrew Poelstra. *On Stake and Consensus*. 2015.

[46] Larry Ren. "Proof of Stake Velocity: Building the Social Currency of the Digital Age." In: *Self-published white paper* (2014), pp. 1–13. URL: http://reddcoin.com/papers/PoSV.pdf.

[47] R L Rivest, A Shamir, and L Adleman. "A method for obtaining digital signatures and public-key cryptosystems." In: *Communications of the ACM* 21.2 (1978), pp. 120–126. DOI: 10.1145/359340.359342.

[48] D Schwartz, N Youngs, and A Britto. "The Ripple protocol consensus algorithm." In: *Ripple Labs Inc White Paper* (2014). URL: http://ripple.com/consensus-whitepaper/.

[49] P. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer." In: *SIAM Journal on Computing* 26.5 (1997), pp. 1484–1509. DOI: 10.1137/S0097539795293172. arXiv: 9508027 [quant-ph].

[50] Yonatan Sompolinsky and A Zohar. "Accelerating Bitcoin's Transaction Processing. Fast Money Grows on Trees, Not Chains." In: *IACR Cryptology ePrint Archive* 881 (2013), pp. 1–31. URL: https://eprint.iacr.org/2013/881.

[51] Global Legal Research Directorate Staff. *Regulation of Bitcoin in Selected Jurisdictions*. Tech. rep. January. U.S. Law Library of Congress, Global Legal Research Center, 2014, pp. 1–25. DOI: January2014. URL: http://www.loc.gov/law/help/bitcoin-survey/regulation-of-bitcoin.pdf.

[52] National Institute of Standards and Technology. "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions." In: *Draft FIPS PUB 202* (2014). DOI: 10.6028/NIST.FIPS.202.

[53] National Institute of Standards and Technology. "Specifications for the Secure Hash Standard - FIPS PUB 180-2." In: *Computing* (2002), pp. 1–71. DOI: 10.6028/NIST.FIPS.180-4.

[54] M Veldhorst et al. "A Two Qubit Logic Gate in Silicon." In: *Nature* 526 (2015), pp. 410–414. DOI: 10.1038/nature15263. arXiv: 1411.5760. URL: http://www.nature.com/nature/journal/vaop/ncurrent/full/nature15263.html.

[55] Marko Vukolić. *The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication*. Tech. rep. IBM Research Zurich, 2014.

[56] Gavin Wood. "Ethereum: a secure decentralised generalised transaction ledger." In: *Ethereum Project Yellow Paper* (2014).

A. Determining success probabilities of a short-range attack

Listing A.1: Python-Code used to calculate the success probabilities of short-range attacks executed on Python version 3.5.1

```
from sympy import Symbol, symbols, Piecewise, init_printing
from sympy.stats import density, Poisson
import numpy as np
import pandas as pd

def density_progress(blocks_behind, p):
    expected_progress = blocks_behind * p / (1 - p)
    return density(Poisson("p", expected_progress))

def overtake_prob(p_network, blocks_behind, required_progress):
    return density_progress(blocks_behind, \
                            1 - p_network)(required_progress) * \
        (1 - ((1 - p_network) / p_network) ** \
          (blocks_behind - required_progress))

def success(p_network, blocks_behind):
    res = 1 - sum([overtake_prob(p_network, blocks_behind, i) \
                   for i in range(0, blocks_behind)])
    return res

def probability_ever_catchup(p, blocks_behind):
    return Piecewise(((p / (1 - p)) ** blocks_behind, p < 0.5), (1, p >= 0.5))

blocks_behind = np.arange(50)
probs = np.arange(0.5, 1, 0.01)

out_probs = []
out_blocks = []
out_res = []

for behind in blocks_behind:
    for prob in probs:
        out_probs.append(prob)
        out_blocks.append(behind)
        out_res.append(success(prob, behind))

d = {'blocks': out_blocks, 'probs': out_probs, 'res': out_res}
pd.DataFrame(d).to_csv('out_probs.txt', sep=' ', index=False)
```

B. Overview of important variants of distributed consensus systems

Table B.1.: Overview of 30 biggest (measured by market cap) implemented distributed consensus systems, sourced: `coinmarketcap.com` on Mar 22, 2017

	Name	Market Cap [$ mn]	Price [USD]	Available Supply [mn]	Volume (24h) [$ mn]
1	Bitcoin	18,166.78	1119.40	16.23 BTC	337.70
2	Ethereum	3,868.46	42.99	89.98 ETH	91,09
3	Dash	686.55	95.64	7.18 DASH	41.50
4	Monero	301.06	21.27	14.16 XMR	18.60
5	Ripple	258.29	0.0069	37,338.11 XRP	3.01
6	Ethereum	215.77	2.40	89.94 ETC	21.18
7	Litecoin	202.41	4.02	50.29 LTC	4.95
8	NEM	141.67	0.0157	9,000.00 XEM	1.17
9	Augur	91.46	8.31	11.00 REP	0.87
10	MaidSafeCoin	81.47	0.1800	452.55 MAID	0.58
11	Zcash	60.77	66.39	0.92 ZEC	6.88
12	Tether	44.95	1.00	44.95 USDT	21.64
13	Golem	41.69	0.0509	820.00 GNT	3.22
14	Steem	37.79	0.1615	233.99 STEEM	0.75
15	Iconomi	33.95	0.3903	87.00 ICN	1.12
16	Lisk	32.43	0.3099	104.66 LSK	6.37
17	DigixDAO	31.90	15.95	2.00 DGD	0.14
18	Factom	31.86	3.64	8.75 FCT	0.77
19	Waves	28.90	0.2890	100.00 WAVES	0.17
20	PIVX	27.19	0.5180	52.51 PIVX	1.20
21	Dogecoin	26.68	0.0002	108,631.31 DOGE	0.24
22	Decred	23.42	5.75	4.08 DCR	0.59
23	Ardor	22.97	0.0230	999.00 ARDR	0.35
24	Melon	18.27	30.49	0.60 MLN	0.13
25	GameCredits	16.26	0.2646	61.45 GAME	0.23
26	Stellar	15.51	0.0021	7,104.81 XLM	0.62
27	Peercoin	14.97	0.6251	23.96 PPC	0.27
28	BitShares	14.67	0.0057	2,590.23 BTS	1.14
29	ShadowCash	13.97	2.10	6.64 SDC	0.96
30	Siacoin	13.77	0.0006	24,329.32 SC	1.41

C. Volatility of Cryptocurrencies

We compare the annualized volatility over a 30-day rolling window of Bitcoin (BTC) and Litecoin (LTC), two cryptocurrencies with a sufficiently long history, to the Euro (EUR) and British Pound (GBP) in figure C.1. Additionally, cryptocurrency volatility is compared to traditionally more riskier assets in the stock and commodity market in figure C.2. We take the Dow Jones as a proxy for the US equity market and the price of gold as a proxy for global commodity markets.

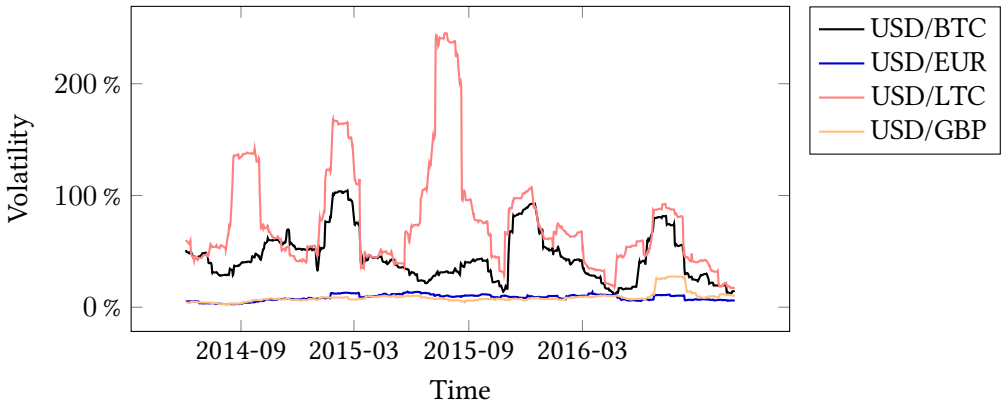


Figure C.1.: Annualized volatility over a 30-day rolling window of different currencies

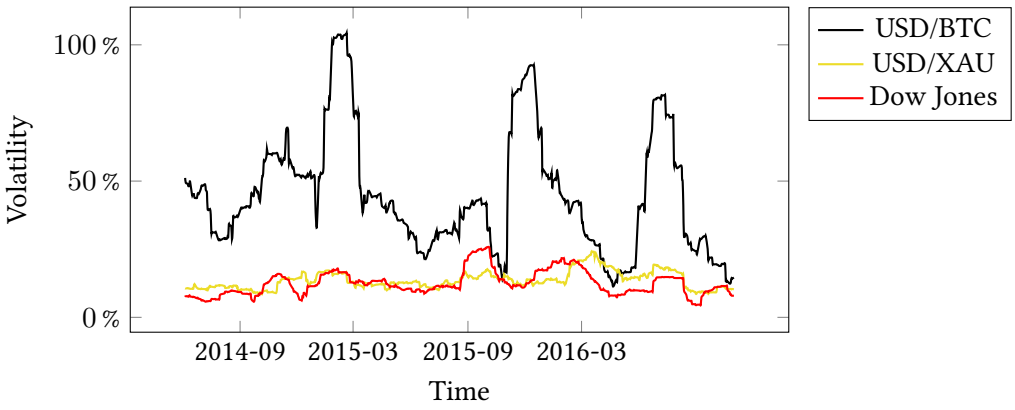


Figure C.2.: Annualized volatility over a 30-day rolling window of Bitcoin compared to other assets

The quantiles of the measured distributions of rolling volatilities are shown in table C.1. We find that cryptocurrency volatility far exceeds that of other currencies or risk assets. Bitcoin and Litecoin feature interquartile ranges of measured volatilities of 31–53% and 47–99%, respectively. In comparison, gold, the next most-volatile asset, features an interquartile range of only 12–15%.

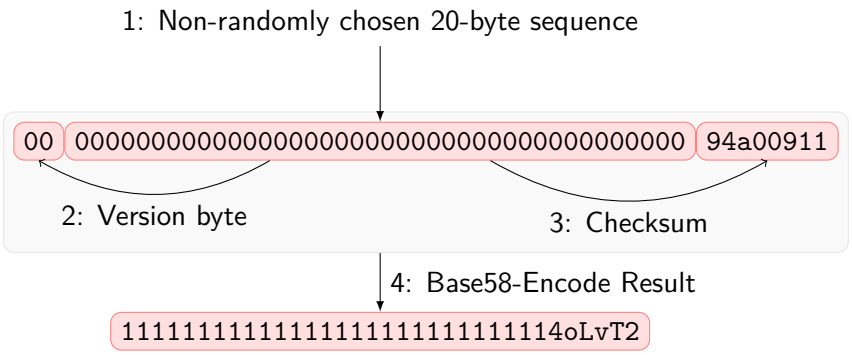
Table C.1.: Distribution of 30-day volatility (in %, scaled to 1 year) of select asset classes (denominated in USD)

Quartiles	BTC	LTC	Gold	Dow	EUR	GBP
0%	11.54	18.92	8.53	5.72	2.58	2.62
25%	30.96	46.91	11.66	9.53	6.94	6.46
50%	41.89	64.33	13.08	11.66	9.31	7.44
75%	53.38	98.75	15.46	15.63	10.87	8.86
100%	103.89	244.99	24.15	25.76	13.99	25.81

D. Burning Bitcoin

In order to burn Bitcoin, coins have to be sent to an address which is not controlled by anyone. Here, we choose a Bitcoin address which corresponds to a public key which hashes to zero (160 bits).

1. Pick an obviously non-randomly generated hash160
2. Add 0-byte to front of byte sequence
3. Calculate checksum (see `checksum()` below; first 4 bytes of doubly-applied `SHA-256` hash function)
   Add checksum to end of byte sequence
4. Base58-Encode complete byte sequence



Listing D.1: bash function to calculate checksum

```
#!/bin/bash
checksum() {
 xxd -r -p <<< "$1" |
 openssl dgst -sha256 -binary |
 openssl dgst -sha256 -hex |
```

```
 head -c 8 |
 awk '{print $1}'
};
```

As a next step, we will burn an amount of 0.0001 BTC by sending it to address `1111111111111111111114oLvT2`.

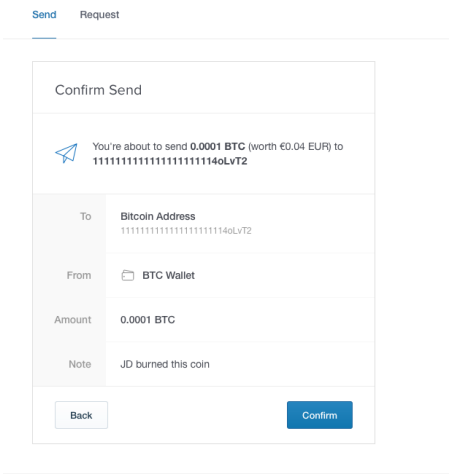


Figure D.1.: Burning Bitcoins by sending them to an unspendable address

The transaction has been commited to the public in the Bitcoin blockchain in block #410973[1].

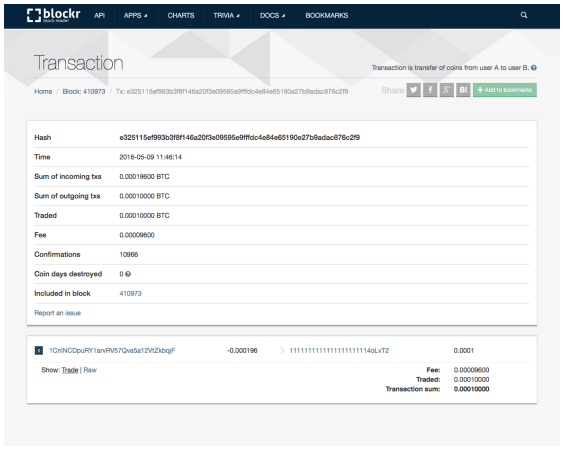


Figure D.2.: Committed transaction in the Bitcoin blockchain

[1] `https://btc.blockr.io/tx/info/e325115ef993b3f8f146a20f3e09595e9fffdc4e84e65190e27b9adac876c2f9`