Alma Mater Studiorum · Università di Bologna

DOTTORATO DI RICERCA IN
COMPUTER SCIENCE AND ENGINEERING
Ciclo XXXV

A Formal Analysis of Blockchain Consensus

Presentata da:
Adele Veschetti

Supervisore:
Prof. Cosimo Laneve

Coordinatore Dottorato: Prof.ssa Ilaria Bartolini
Settore Concorsuale: 01/B1 - Informatica
Settore Scientifico Disciplinare: INF-01 - Informatica

Esame finale anno 2023

A Brenno e Margot.

Abstract

Blockchain systems are distributed ledgers that records transactions on multiple computers. Each blockchain system uses a consensus protocol to secure the network and validate transactions. Bitcoin is the most famous blockchain system whose consensus protocol is the proof of work. Hybrid Casper blockchain is a proposed improvement of the Ethereum blockchain that uses a combination of proof-of-work and proof-of-stake as consensus protocol.

In this thesis, we analyse these protocols using PRISM+, our extension of the probabilistic model checker PRISM with blockchain types and operations upon them. This allows us to model the behaviour of key participants in the protocols and describe the protocols as a parallel composition of PRISM+ processes.

Through our analysis of the Bitcoin model, we are able to understand how forks (where different nodes have different versions of the blockchain) occur and how they depend on specific parameters of the protocol, such as the difficulty of the cryptopuzzle and network communication delays. Our results corroborate the statement that considering confirmed the transactions in blocks at depth larger than 5 is reasonable because the majority of miners have consistent blockchains up-to that depth with probability of almost 1. We also study the behaviour of the Bitcoin network with churn miners (nodes that leave and rejoin the network) and with different topologies (linear topology, ring topology, tree topology and fully connected topology).

PRISM+ is therefore used to analyse the resilience of Hybrid Casper when changing various basic parameters of the protocol, such as block creation rates and penalty determination strategies. We also study the robustness of Hybrid Casper against two known attacks: the Eclipse attack (where an attacker controls a significant portion of

the network's nodes and can prevent other nodes from receiving new transactions) and the majority attack (where an attacker controls a majority of the network's nodes and can manipulate the blockchain to their advantage).

The thesis includes the analysis of related works and elaborates on possible future research directions.

Acknowledgements

First and foremost, I want to begin by expressing my gratitude to my supervisor, Cosimo Laneve, for consistently making himself available to engage in discussions and provide guidance regarding our research, as well as my personal life.

I am also grateful to Einar Broch Johnsen and Massimo Bartoletti for spending time on reviewing this thesis.

In the last four years, I shared wonderful moments with my colleagues and friends. So, a special thanks goes to Luca, Angelo, Federico, Francesco, Ivan, Andrea, Ken, Delfina. Thank you for all the lunch breaks, the aperitivi and the uncountable coffees.

I would like to highlight also the exceptional people from TU Darmstadt. A special thanks to Reiner Hähnle and Richard Bubel for hosting me during my period abroad and for giving me the opportunity to be a part of their project. Thanks also to my german colleagues and friends. You have always made me feel so welcomed, without you Darmstadt would not feel like home.

I also need to thank my incredible friends that supported me throughout this journey. Thanks to Vittoria. Even if we have been long distance friends for many years, I know that you are always there for me, on Facetime, ready to share both the joy of the beautiful moments and the weight that sometimes life makes us carry on our shoulders.
Thanks to *i Balotti*, lifelong friends. They have been my constant through all this.

I would also thank Mirella and Daniele for loving me as their daughter.

Finally, but the most importantly, I would like to thank my family. Thanks to my mother, Emanuela, for being the strongest woman I have ever met. Thanks to my little sister, Diana, who is my biggest cheerleader and who is a pillar of strength to me when life gets harder. Thanks to my father, Brenno, who was not able to see the end of this

journey, but taught me the importance of studying hard and dedicated his whole life to ensure us a better future. Thanks also to Reki and Margot, who never left me alone.

My final thank goes to my husband Alessandro, I would not have finished my PhD without him. Thank you for being my strength, for pushing me when I needed it and for always supporting my decisions. Thank you for making me laugh no matter what.

Contents

Symbols

r_b	exponential parameter modelling communication latency of the network
mR	exponential parameter modelling the time needed to create a block
lR	exponential parameter modelling the time needed to not create a block
hR_i	hashing power percentage owned by the i-th node
s	seconds needed to create a block
rC	exponential parameter modelling how often a validator should vote
$r_{i,j}$	rates for the time needed by a churn node to leave/join the network
N	number of nodes
T	bound time for experiments (seconds)
γ	percentage of the increased/decreased stake
len_{epoch}	epochs length
$P_{k_{fork}}$	probability of a fork of length k
P_{k_just}	probability that a checkpoint is justified within k epochs
P_{k_fin}	probability that a checkpoint is finalised within k epochs

Part I

Background

Chapter 1

Introduction

Blockchain has revolutionised the way individuals and companies exchange digital assets without the control of a central authority. This technology has been successfully exploited in different contexts, e.g., the management of cryptocurrencies (Bitcoin being the most famous one [62]), running decentralised applications (Ethereum smart contracts [18]), the implementation of voting systems [14], and Decentralised Finance [64].

Blockchain's main novelty is to enable a dynamic and asynchronous network of peer-to-peer nodes to maintain a distributed ledger that globally records the occurrence of certain events. Nodes contain a local copy of the ledger that is updated upon reception of special messages, called *transactions*. Due to the inherent asynchrony of the network, the main difficulty that a blockchain system must address is the consistency of the ledger upon updates performed by different nodes. In particular, it may happen that two or more ledgers differ and this situation is called a state of *fork*. To overcome this problem, which has been demonstrated unsolvable in 1985 [36], these systems rely on consensus protocols with probabilistic approaches. Traditionally, following the seminal work by Nakamoto [62], these protocols have been based on a probabilistic mechanism called Proof of Work (PoW) whereby nodes can update the ledger by creating new blocks – *mining* – only if they solve a moderately hard computational problem. Additionally, to further reduce the probabilities of inconsistencies in the copies of the ledger, Bitcoin guarantees the so-called *eventual consistency* whereby the various replicas may be temporarily inconsistent in at most the last m blocks. In particular, Bitcoin considers as

"*confirmed*" (and therefore "*can be paid*") every block that is at depth greater than 5 [42].

The Bitcoin protocol is complex and many researchers are actively involved in studying its properties and its criticalities. Clearly, understanding the details of the protocol is of paramount importance because overlooking some of them might introduce vulnerabilities and pave the way to attacks. For example, forks may be used for rewriting the transaction histories and for letting the blockchain evolve to a wrong state. A typical example of a wrong state is a state where a transaction is paid twice – called a *double spending attack*.

However, because of the hardness of the computational problem, PoW has the substantial shortcoming of requiring a very large amount of computational resources and energy [3]. For this reason, new proposals have been emerging, the most popular being Proof of Stake (PoS) where nodes can update the ledger with a probability that is proportional to the quantity of cryptocurrency they invested to be part of the network – the *stake*. One of these protocols – the Casper Protocol [19] – has already been adopted by Ethereum. Before the adoption, to ensure a smooth transition with minimal impact on the users, Ethereum developers deployed a hybrid version of Casper – the *Hybrid Casper Protocol* – that uses both PoW and PoS [21]. In particular, Hybrid Casper speeds up block creation through a less expensive PoW than Bitcoin (block creation occurs every 14 seconds in Hybrid Casper [21], while it takes 600 seconds in Bitcoin [27]) and uses a voting mechanism to select the blocks to append to the blockchain.

Votes are expressed by suitable nodes of the network that own a *stake*, called *validators*, and for certain blocks, called *checkpoints*.[1] These checkpoint blocks pass through two stages: the first is when the checkpoint is *justified*, which means that it has received at least 2/3 of the validators' votes in terms of stake; the second is when it is *finalised*, which means that it is justified *and* its child checkpoint is justified as well. Finalisation guarantees the consistency of the corresponding blockchains in the distributed ledger.

In this thesis, we analyse the probabilistic behaviour of Bitcoin and Hybrid Casper by using formal methods.

Following the approaches of [68, 42], we use an abstract model that defines the

[1] Checkpoints are blocks whose block number/height is a multiple of 64, which is called *epoch* length.

network of nodes as the *parallel composition* of processes and the time needed to mine a block and to broadcast a message as *exponential distributions* with a *rate parameter* associated to process actions. As done in [12, 33, 85], we use Continuous Time Markov Chains (CTMCs) for providing a probabilistic model of our processes.

We started our research by defining the Bitcoin model through a stochastic calculus and, by analysing the transition system, we computed the probability of having a fork of length m. We have therefore computed the probability of devolving into a "larger inconsistency", e.g. transiting from a state with a fork of length m to a state with a fork $m+1$ and to formulate a theorem stating how this probability changes. We also applied the same technique for studying an attack to Bitcoin that has been already discussed in [62]: the presence of hostile nodes mining new blocks in positions that are different from the correct one (blocks are not inserted at maximal depth). The foregoing work required a time-consuming analysis of the stochastic transition system and, for this reason, we decided to model Bitcoin by using a model checker to perform automatic analyses and of examining the correctness of the protocol in different settings. As stochastic model checker, we committed to `PRISM` [52] because it allows us to perform stochastic model checking and, in particular, to analyse CTMC models. Actually, in order to have a more precise encoding, the model checker we considered is an extension of `PRISM` with blockchain data types – `PRISM+`. The software package is available online [38] and can be used to model and study the quantitative properties of generic PoW and PoS protocols.

We, then, modelled the Bitcoin protocol in `PRISM+` as a parallel composition of n miner processes, n hasher processes and a process network. As an initial step, we assess the coherence of our model by verifying that the probabilities of mining a new block within a given amount of time and the probability of having a fork are both in full agreement with the values available from the literature [27]. We also study the trade-off between the security guarantees and the difficulty of the cryptopuzzle. More precisely, we analyse the variability of the probability of reaching fork states when the speed of the mining process increases, that is when the difficulty of the cryptopuzzle decreases. We also study a network with churning nodes, i.e. nodes that can leave and rejoin the network. In particular, we analyse how the presence of this kind of node affects the probability of mining new blocks and, consequently, the probability of forks. Moreover,

we consider several network topologies (linear topology, ring topology, tree topology and fully connected topology) and analyse them in order to estimate how the connections between nodes can affect the likelihood of a fork.

Furthermore, we modelled Hybrid Casper as a PRISM+ process and verified that the model was compliant with the results shown in [21] when the rate of actions are the same. We give a stochastic characterisation of the safety and liveness properties proposed in [21] and we verify that they hold even when changing the time needed to deliver a block. We verify that increasing the rate of creation severely impacts on the justification/finalisation of blocks and we analyse different penalty strategies, by studying different quotas of penalty in case of misbehaviours of validators. Finally, we compute the probabilities of misbehaviours of Hybrid Casper against two well known attacks. First, we consider *the Eclipse attack*, where an adversary obstructs the delivery of messages to some nodes of the network, and force them to work on an untruthful view of the blockchain. Then, we focus on *the majority attack* where an attacker (or a coalition of attackers) controls the majority of the network and works on creating a separate blockchain.

1.1 Research Method

The main goal of my research has been to model and analyse the main blockchain consensus protocols. The idea was to formalise the main properties of the selected consensus protocols and their vulnerabilities and, by means of a probabilistic analysis, to verify them. The general properties we decided to study are:

(i) which is the probability of reaching a state of fork;

(ii) how this probability changes by varying some fundamental parameters (e.g. the latency of the network, the time needed to create a new block, etc.).

We analysed the Bitcoin protocol and the Hybrid Casper protocol, and it is worth noticing that, in both the models, we suppose that

(i) *the proof of work is negligible*: we model the overall effect of creating a new block through an ad-hoc action and we ignore the algorithmic process of mining;

(ii) *blocks are black boxes*: we omit any informations that is not relevant for the consensus, such as transactions.

This has been possible because of we wanted to study properties that were not influenced by these aspects.

Moreover, in order to abstract out the solution of the cryptopuzzle of the proof of work and the time needed to deliver new blocks within the network we use exponentially distributed rates, obtaining Continuous-time Markov chains models. This is in line with the literature:

(i) the time spent by a node i to mine a block in a proof of work blockchain can be described by an exponential distribution $1 - e^{-\lambda_{m_i} t}$, where the parameter λ_{m_i} depends on the miner hashing power and the difficulty level of the crytopuzzle (see Nakamoto [62]);

(ii) the communication delay across the network can be also approximated by an exponential distribution [27].

Comparison With Other Approaches

Assessing the security of a distributed system is a fundamental activity, which is even more stringent for blockchain systems that manage crypto-assets of a high economic value. Since the security of such systems strictly depends on the consensus protocol that is used, it is essential to assess which properties this protocol enjoys. As already explained, we decided to analyse the main blockchain consensus protocols by using a formal verification approach. In particular, the main goal was to detect potential vulnerabilities of the models we considered and being able to analyse different settings by changing few parameters. In this section we overview the *automatic techniques* that have been proposed in the literature and position our approach. Additional details can be found in Chapter 8. There are three mainstream approaches for *automatic* analysing consensus protocols: testing, simulation and formal verification.

Testing. In this approach, the system under test runs in a virtual network or a simulated environment under varying configurations that resemble as much as possible the

production environment. Usually, the goal is to evaluate how the system behaves under different values of the parameters such as network conditions, workloads, and attacks. To perform this evaluation, testers require generating the network traffic, simulating the attackers, and implementing mechanisms that measure the properties of interest. For example, the test net used in Buterin [21] tests the behaviour of Ethereum protocols in scenarios that are similar to the final one. It is frequent that test nets spot bugs, but it also happens that bugs may remain uncaught and displayed by the final system. The testing approach usually imposes a severe burden on testers that have to set up an actual distributed infrastructure, generate the relevant network traffic, and simulate the attackers. The deployment of a large-scale distributed computing test net is often tedious, time-consuming, and costly. For these reasons, testers hardly reproduce a precise deployment environment due to limited financial and timing resources.

Simulation. The second approach uses simulators [69], which implement the protocols by ad-hoc modules that try to reconstruct the overall behaviour on a single machine [35]. These implementations rely on *simulation models*, which are stochastic in the case of blockchains, *e.g.* continuous Time Markov Chain, Markov Decision Process, etc. Blockchain simulators allow designers to reproduce real-world processes in a low cost manner, such as network latency and bandwidth. Additionally, by changing parameters of the simulation, the system can be analysed without the need to re-implement it. So, simulators allow users to quickly test a blockchain system using different settings and parameters, to study its behaviour under various operational scenarios and to choose the proper configuration settings. For example, Gervais et al. [40] introduce a quantitative model based on Markov chains to compare PoW blockchains. The model allows them to reason about optimal adversarial strategies while taking into account the adversarial mining power, the impact of eclipse attacks, block rewards, and real world network and consensus parameters. The system is however different from the original implementation and simulations only highlight particular executions. In general, the development of simulators is complex. Most simulators can realistically reproduce only one or few aspects of the (blockchain) system leaving the other ones simplified, or even skipped entirely.

Automatic Formal Verification. The third approach, which is the one implemented in our technique, for verifying distributed protocols relies on formal verification using an automatic tool, therefore its application requires no supervision or expertise in mathematical reasoning and covers almost all possible behaviours of the system. Among the various techniques, model checking has been widely applied to consensus protocols [29, 54, 83, 53]. With respect to testnets, model checking has the advantage that it is relatively cheap (no network infrastructure nor the relevant network traffic is needed to be generated) and it is relatively fast to stress-test the protocol under different settings and conditions because it suffices to adjust model's parameters. With respect to simulations, model checking has the advantage to undertake a (more) complete analysis of the possible executions.

However, model checking has some drawbacks. The first one is that one analyses an abstract model rather than the actual implementation of the protocol. Thereforesome precision is necessarily lost. Additionally, the definition of the abstract model takes time since it is essential to understand the modelling language and the protocol (in our case, the process of defining the model took us around a couple of weeks). The second drawback is that the analyses are time-consuming due to the state explosion problem (the whole model, or an approximation of it, must be completely generated). For example, in our experiments, verifying a network with eight nodes takes around four hours, while it takes around seven days when the nodes are sixteen. In particular, to bound our analyses, we ran the experiments till the results stabilise, which occurred when nodes are in between 12 and 16. (For this reason, 16 has been the maximal size of our networks). The third one is that, to further reduce the state explosion, one resorts to approximations of model checking, such as the so-called statistical model checking that compromises testing and classical model checking techniques, which will be presented in Chapter 3.2.

Overall, we think that every automatic analysis approach has pros and cons. When using testing in a toy scale real scenario, we know we will be able to spot bugs that would be impossible to find with any other approach. At the same time, we have to accept the fact that even this scaled down scenario would require a huge computational effort.

When using simulation, we have the power to truly recreate and represent even the tiniest details of the network (e.g. the network latency), but, on the other hand, it would

be hard to claim that a simulation could faithfully represent the system as a whole.

When using formal verification, and in particular model checking, we clearly highlighted the unique possibility to formally prove a specific and interesting property of the system. Meanwhile, we have to admit that given the necessity to reproduce all the states of evolution of the system, the required time to explore all the possibile states can be really burdensome.

Therefore, we can claim that even in our field of study, a deep and comprehensive study of blockchain consensus protocols, there is not such a thing as one-size-fits-all approach. However, it is reasonable to use them all together. Clearly, this integrated approach is yet to be implemented, but in light of the studies mentioned in this section and the ones presented in Chapter 8, we can say this would be useful and desirable. An integrated usage of the previously mentioned techniques would lead us to the opportunity to spot a large number of bugs at the early stages of software development. In this view, we think that our approach adds a new axis to the analysis of blockchain protocols that may complement the other techniques. Finally, we think that this integrated approach would be of major interest given that, as we discussed before, pros and cons of these methods are mostly complementary. Therefore, using them together would grant us to get the best out of each.

1.2 Outline of the Thesis

The thesis is organised as follows. Chapter 2 is an overview about the blockchain system and the consensus protocols, while Chapter 3 provides a quick introduction about model checking and the model checker `PRISM`. The description of the `PRISM+` extension with the data types and the operations is detailed in Chapter 4. The Bitcoin `PRISM+` model is presented in Chapter 5. The chapter also presents the analyses made for the Bitcoin protocol. Chapter 6 shows the results obtained by analysing the Bitcoin protocol considering of the stochastic transition system of the model, discussing both the results obtained in a system with honest nodes and the ones obtained in presence of an attack. The Hybrid Casper model is detailed in Chapter 7, we also present the results obtained testing our model. Chapter 8 compares our proposal with the literature and

gives a general overview of the techniques used to analyse blockchain protocols and in Chapter 9 we draw some conclusions and discuss possible future works.

Chapter 2

Blockchain Systems

Blockchain is revolutionising the way individuals and companies exchange digital assets without the control of a central authority. It has been introduced in 1990 by Haber and Stornetta [43] and its main novelty is to enable a dynamic and asynchronous network of peer-to-peer nodes to maintain a distributed ledger that globally records the occurrence of certain events. Since its introduction, it has evolved to become a powerful tool for securely and transparently recording transactions in a decentralised network.

One of the key features of blockchain is that it is a distributed ledger technology, meaning that it is maintained by a network of peer-to-peer nodes rather than a central authority. This allows for a high degree of autonomy and decentralisation and can reduce points of weakness in systems where there may be too much reliance on specific actors. Another important aspect of blockchain is that it enables the recording of events in a global and asynchronous manner, meaning that transactions can be recorded at different times and locations and still be registered on the ledger. Blockchains can be applied in a wide range of applications, from financial transactions to supply chain management and digital identity verification. This section provides an overview of blockchain technology and the most commonly used consensus protocols in the field.

2.1 A Blockchain Overview

A blockchain is a *distributed database that maintains a continuously growing list of ordered records, called blocks*, connected through cryptographic functions. Each block in the blockchain is identified by a hash, created using the SHA256 algorithm, and contains a reference to the previous block in the form of the "previous block hash" field in the header. This creates a chain of blocks linked by hashes, with the first block being the *genesis block*. This means that if the parent block is modified in any way (such as by changing the transactions stored within it), the parent's hash will also change, and the "previous block hash" pointer in the child block must also be updated accordingly. This change to the child block's hash will then require the grandchild block's hash to be updated, and so on, creating a cascade effect where all subsequent blocks must be recalculated. This makes it extremely difficult to alter the deep history of a blockchain, as it would require a massive amount of computation to recalculate all of the subsequent blocks. This immutability of the blockchain's history is an important aspect of its security.

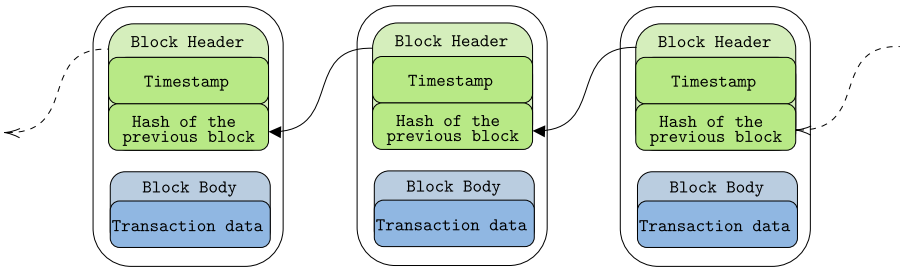


Figure 2.1: A simplified example of how blocks are chained to form a blockchain.

Each block also stores a timestamp and transaction data, generally represented as a *Merkle tree*. Merkle trees are binary trees containing cryptographic hashes and they are used for efficiently summarising and verifying the integrity of large sets of data. Each block contains a *timestamp*, a unique serial number, which is used to determine the exact moment when the block was mined and validated by the network. It ensures that the transaction data existed at the time the block was created.

In this thesis, a ledger is denoted as $\mathtt{L}$, $\mathtt{L'}$, ... and formally defined as follows.

Definition 2.1. *The ledger datatype is a pair* $\mathtt{L} = \langle \mathtt{T}; p \rangle$ *where* $\mathtt{T}$ *is a tree of blocks and* p *is the pointer to the last valid leaf block, called the* handle *of* $\mathtt{L}$.

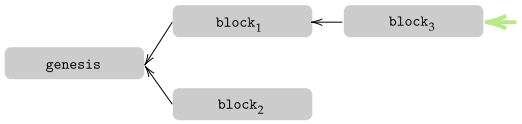


Figure 2.2: A ledger and the handle in green arrow.

Essentially, the handle represents what individual nodes accept as the last valid block, which varies depending on the protocol. For the purpose of this thesis, which concentrates on proof of work protocols, the handle indicates the leaf block located at the greatest depth. Worth of notice is that, with this definition, for any block on the chain, there is exactly one backwards path from the handle to the genesis block. The chain of valid blocks (i.e. those blocks whose transactions will be payed) – the blockchain – is the chain from the handle to the genesis block. Formally:

Definition 2.2. *Given a ledger* $\mathtt{L}=\langle \mathtt{T}; p \rangle$, *the* blockchain *of* $\mathtt{L}$, *noted* $\mathtt{L}\uparrow$, *is the sequence* $(p_0, p_1, p_2, \dots)$ *such that* p_0 *is the handle of* $\mathtt{L}$. *For every* i, p_{i+1} *is the parent of* p_i *and* p_n *is the genesis block.*

In Figure 2.3, the blocks in the blockchain are those in blue. The blocks that are not included in the blockchain (grey) are considered garbage.

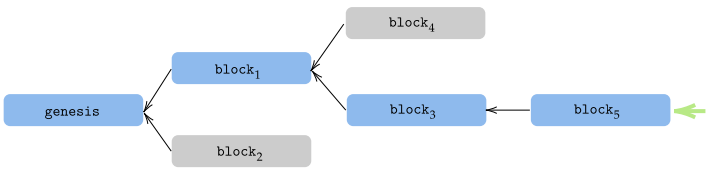


Figure 2.3: A ledger and the corresponding blockchain (blue blocks).

2.1.1 Blockchain Forks

Because the blockchain is a distributed data structure, it may happen that different copies of it are not always consistent. The condition of having different blocks at the same height is called *(temporary) fork*. In a blockchain, a fork occurs when different blocks are discovered almost simultaneously by different nodes, and multiple child blocks temporarily refer to the same parent block. As a result, the network can temporarily split into different branches, with each branch containing a different valid block.

During a fork, only one of the child blocks will eventually become part of the blockchain, while the others will be discarded. The process of resolving a fork and choosing which block becomes part of the blockchain depends on the type of protocol being used. In a proof-of-work protocol, such as the one used by the Bitcoin blockchain, the block with the most proof-of-work (i.e., the block that required the most computational effort to produce) is typically chosen. In a proof-of-stake protocol, such as the one proposed for the Hybrid Casper blockchain, the block chosen may be based on the stake (i.e., the amount of cryptocurrency held by the node) of the node who produced it. Moreover, some blockchain consensus protocols guarantee fork free systems [57]. Overall,

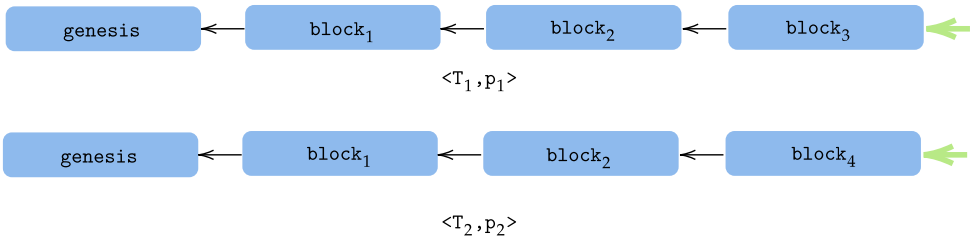


Figure 2.4: Two ledgers in a state of fork of length 1.

forks are a normal part of the blockchain process and are resolved through the protocol's consensus mechanism. They allow the blockchain to continue functioning even when there are temporary disagreements about the state of the network. However, on the other hand, forks may be used for rewriting the transaction histories and for letting the blockchain evolve to a wrong state. A typical example of a wrong state is a state where a transaction is paid twice – called a double spending attack. Formally, it is possible to define a fork as follows.

Definition 2.3. *Let* $L_1 = \langle T_1; p_1 \rangle$ *and* $L_2 = \langle T_2; p_2 \rangle$ *be two ledgers and let*

- m_1 *be the length of* $L_1 \uparrow$,
- m_2 *be the length of* $L_2 \uparrow$,
- h *be the length of the maximal common suffix of* $L_1 \uparrow$ *and* $L_2 \uparrow$.

We say that L_1 *and* L_2 have a fork of length k, *where* $k = \max(m_1 - h, m_2 - h)$.

To better understand Definition 2.3, consider the two ledgers in Figure 2.4. These ledgers have a fork of length 1 because they differ in the handles, i.e., the blocks pointed by the green arrow.

2.1.2 Blockchain Classification

Blockchain systems can be classified into two main categories based on permissions: *permissioned* and *permissionless*.

- Permissionless blockchains, also known as public blockchains, do not require explicit permission for participation. This kind of network is open to anyone, allowing them to participate in various activities such as sending and receiving transactions, operating a node, viewing and contributing to the code, and taking part in the consensus process. Examples of permissionless blockchains include Ethereum [32] and Bitcoin [15].

- Permissioned blockchains have restricted access, controlled by a specific entity or group. Access to these networks is granted through permission and participants are usually required to pass a screening process before being allowed to participate in the activities of the network. Permissioned blockchains are often used in private or consortium settings, where the focus is on increased security and control over who can participate in the network. An example of a permissioned blockchain is Quorum [74].

There are also *public-permissioned* blockchains. These networks combine the permissioning from private consortiums with a decentralised governance model, trying to achieve the best properties of both models.

Blockchains can also be classified based on other parameters such as their application domain (e.g., financial, supply chain, healthcare), the protocols they use (e.g., proof-of-work, proof-of-stake), and their consensus mechanism (e.g., centralised, decentralised). Other kinds of classifications are not covered since they are not relevant to our purposes.

2.2 Blockchain Consensus Protocols

A consequence of the fact that a blockchain is a distributed digital ledger used to record transactions is that these transactions need to be verified by nodes participating in the network before they can be added to the chain. This system helps to protect the blockchain from fraudulent activity by its peers.

To ensure that all participants (*nodes*) in a blockchain network have the same understanding of the history, blockchain networks use a *consensus mechanism*. This is a set of protocols that enables a network of nodes to agree on the state of the blockchain. There are several types of consensus protocols, such as *proof of work*, *proof of stake*, *proof of authority*, etc.
In the next sections we will discuss the protocols we will analyse in the remaining part of the thesis: the Bitcoin proof of work protocol [62], the Proof of Stake protocol and the Hybrid Casper protocol [21], which is an hybrid version between the first two.

2.2.1 Proof of Work

In a proof of work blockchain the nodes in the system are called *miners*. According to PoW, miners can add a block only if they solve a computational problem. Technically, the problem consists of finding a number (a nonce) that is inserted into the block header. The block header is then hashed and if the numerical value of the hash is less than a predefined condition, which is called target, then the miner is said to have mined the block. The only way to find such a nonce is through an exhaustive search. The finding

of suitable nonce values can be modelled as a Bernoulli trial with a probability of $L/2^{256}$ of being successful, where L is the target. The time needed to mine a new block depends on the difficulty of the PoW and the hashing power of the miners. Clearly, the faster miners are, that is the more computational power they own, the higher the probability of forks and, thus, the more likely the inconsistency between miners.[1]

The Bitcoin Protocol

Bitcoin uses the proof of work consensus protocol. As explained by Nakamoto [62], the time it takes miner i to solve a block problem is exponential with parameter λ_{m_i}. The parameter λ_{m_i} is determined by the miner's individual computing power, or hash power, hR_i, and by the difficulty of the mining task set by the network protocol, D. Thus, one can obtain:

$$\lambda_{m_i} = \frac{hR_i}{D}$$

In this work, we will consider the parameter mR as $\frac{1}{D}$, thus, the time needed by the i-th miner to create a block will be approximated by the exponential parameter

$$\lambda_{m_i} = hR_i \times mR.$$

The difficulty is set by the protocol so that the expected time between two block solutions is targeted to a constant, in Bitcoin this time is set around to 10 minutes, which means that the parameter mR can be set to 600 seconds.

Once a block has been mined, the miner (i) adds the block to its own ledger (therefore the depth of the ledger increases and the handle is updated); and (ii) broadcasts it to all the connected nodes of the network. Every node receiving the new block updates its local copy of the ledger by inserting the block in the right position and, if necessary, it also updates its own handle. If the block cannot be connected to the ledger (because, due to network delays, a previous block has not been delivered) then it is added to the local set of the miner and will be inserted afterward (orphan blocks).

As previously mentioned, because of asynchrony, it may happen that two nodes

[1]The probability of a fork is affected also by other factors such as the block size, the latency of the network and the difficulty of the puzzle.

mine and broadcast a block almost concurrently, yielding different ledgers with different handles, i.e. a situation of a fork. This phenomenon is at the core of the inconsistencies in Bitcoin and, proof of work is used in order to overcome this problem. For this reason, the Bitcoin PoW difficulty is determined by a moving average targeting a certain number of blocks per hour. If they are generated too (slowly) rapidly, the difficulty is (decreased) increased as shown by Nakamoto [62]. The current protocol modulates PoW in order to have 6 blocks per hour on average. To further reduce the probability of inconsistencies, Bitcoin also uses the so-called *eventual consistency* (also known as n-consistency [68]). This is a weak version of consistency, according to which the protocol considers consistent those ledgers with the corresponding blockchains equal up to the last few blocks. In particular, Bitcoin considers both transactions and miners' rewards in blocks at a depth greater than 5 as confirmed [5]. This number is called *confirmation* and represents the number of blocks that have been added to the blockchain after a specific transaction has been made and has been accepted by the Bitcoin network. However, it may still happen that there are several leaf blocks at maximal depth. In this case, in Bitcoin, a miner chooses the valid block as the first received leaf at maximal depth, precisely following the description of the protocol presented by Nakamoto [62].

2.2.2 Proof of Stake

PoW comes together with an enormous energy demand [55, 37]. This has made blockchain enthusiasts and researchers think of alternatives for PoW.

One of the main candidates between the proposed alternatives is the *Proof of Stake* (PoS) in which validators (the nodes that can create the next block) are chosen based on the amount of cryptocurrency they hold or "stake" in the network. The more cryptocurrency a validator has staked, the more likely they are to be chosen to create new blocks and earn transaction fees. In the simplest case, stake is the amount of currency, but it can also be (for example) the age of the coin that a node holds. The validator receives the opportunity to append to the branch that they select a new block and simultaneously collect a block reward.

This protocol succeeds in reducing energy expenditure to negligible levels and there are several mainstream blockchains that feature PoS consensus protocols [41, 70, 16, 66].

The Hybrid Casper Protocol

Ethereum [18] is one of the most famous blockchain systems and has switched from the PoW to the PoS protocol in September 2022. In order to do so and to ensure a smooth transition with minimal impact on the users between the two, implemented an hybrid version using both the PoW and the PoS mechanisms.

Hybrid Casper [21] is a protocol for Ethereum that keeps the ledgers consistent by using two consensus techniques: it exploits PoW as block proposal mechanism and PoS to choose a stable blockchain. As usual in PoW, nodes have to solve a computational problem to add new blocks, whose difficulty is set so that a solution is found within 14 seconds.[2] Additionally, the protocol uses a voting mechanism to select the blocks to append to the blockchain. We overview the protocol by highlighting the main features in different paragraphs.

The Hybrid Casper smart contract. The PoS protocol is implemented through a *special smart contract* stored on the Ethereum blockchain that records the current set of active nodes and manages their stakes and the voting process. The nodes of the network that own a stake are called *validators* and they can vote for certain blocks. In particular, nodes willing to become validators create a stake by locking 32 ether (the cryptocurrency of Ethereum), which is performed by calling a deposit function of the smart contract. Conversely, a validator may exit from the active validator set by invoking a logout function (validators need to wait a minimum period after depositing before being allowed to withdraw).

Justifications and finalisations. The goal of the voting process is to *justify* and *finalise* checkpoints, which are blocks whose height is multiple of an *epoch*[3] in the ledger: a checkpoint is justified if it is voted by validators that own at least 2/3 of the stake of the overall network; when two consecutive checkpoints are justified, the older one becomes finalised. (The root block of the ledger, the genesis block, is both justified and

[2] See the GitHub implementation at `https://github.com/ethereum/eth2.0-specs`

[3] An epoch is the contiguous sequence of blocks between two checkpoints, including the first but not the latter. We denote the length of an epoch with len_{epoch} and we set it to 64 as in the GitHub implementation.

finalised by definition.) This mechanism ensures that a finalised block together with all its ancestors belong to the valid chain, and thus, can be considered as permanent (and the transactions linked to the block are permanently recorded on the blockchain). Once a checkpoint is finalised, the validators are paid, and their payment is proportional to the deposited stake.

The fork choice rule. During the vote, validators follow *the fork choice rule* to select the next checkpoint: *the next checkpoint is the block at maximum height that the validator received first.* When a set of validators are *incorrect*, that is deviate from the protocol (e.g. the chosen block is not the one at maximum height that has been received first or more than one block is voted) a *fork* between different justified checkpoints may occur.

Penalties. To prevent validators from misbehaving, the protocol relies on economic incentives and penalties: validators who voted correctly during an epoch are rewarded, while validators who did not are penalised. This is achieved by adjusting validators' stakes according their own voting behaviour: when a checkpoint is finalised the stakes of validators who voted for it is increased by a positive interest rate r (see Section 7.3 for details), whereas the stakes of validators who voted for other checkpoints are shrunk. The penalties grow in proportion of non-voting validators. If epochs fail to be finalised for a long time, the penalties become more and more severe. When a validator engages in clear misbehaviour, *e.g.* by voting for conflicting checkpoints, then its deposit can be reduced as a form of punishment. Incorrect votes are not punished as harshly as conflicting votes, as there are protocol behaviours that can cause a validator to fail to produce a valid vote. Note that the deposits of validators are updated only after checkpoints get finalised.

Properties of Hybrid Casper. Standard attacks to PoS protocols include the nothing-at-stake attack and the class of long range attacks, such as the posterior corruption [23], which will be all detailed in Section 2.3. In the nothing-at-stake, the attacker generates multiple conflicting blocks to maximise its benefit without risking its stake. This kind of attack is irrational in Hybrid Casper by design because of its penalisation mechanism: misbehaving validators are discouraged to generate conflicting blocks by the loss of stake. In the posterior corruption attack, the attacker creates a new branch that originates from

an earlier block, with the goal of surpassing the main blockchain and rewrite the history of the blockchain. In Hybrid Casper the blockchain up to the most recently finalised checkpoint will never be reverted guaranteeing that a posterior corruption attack can not be successful. In simple terms, a revision fork that finalises blocks older than the last finalised block will be ignored, because all clients will have already seen a finalised block at that height and will refuse to revert it. Another assumption made is that each client will log in the system and gain a complete view of the updated chain at some regular frequency [19].

2.3 Blockchain Attacks

While blockchain technology is often touted for its security, privacy, and immutability, it is not immune to attacks. Several types of attacks can be launched against blockchains. In this section, we will discuss the principal attacks for both proof of work and proof of stake blockchains.

Double spending is a critical issue that blockchain technology aims to address since its inception. It refers to the scenario where an attacker attempts to spend the same digital currency multiple times by first making a transaction, waiting for it to be approved, then reversing it and spending the same currency in another transaction. This is not possible with physical currency, as it is a physical object, and can only be spent once. In a blockchain, double spending can be achieved by presenting a conflicting transaction, possibly in a different branch, to the network.

A Sybil attack is a type of attack in which an adversary creates and uses multiple identities to manipulate a network's decision-making process or influence its opinion. In the context of blockchains, this can lead to various problems such as finalising a block (for example in Hybrid Casper), branching the blockchain, or electing validators.
For a PoW system, the attacker would have to own various identities with sufficient computational power to have substantial influence in the system. The consensus protocol directly defeats this argument as the attacker would have to split his computational power in smaller pieces. This would not be a benefit for the attacker because an attacker's computing power will not be increased by distributing it among multiple identities. In

fact, it will either stay the same if all identities mine the same block or decrease if each identity mines a different block. The attacker will still need to have a significant amount of computing power, regardless of whether it is concentrated in one entity or spread among multiple ones.
Similarly, in PoS systems, the attacker is also hindered by the consensus protocol, as the validator selection process and voting power are determined by the amount of stake held. The attacker's voting power will not be increased by distributing their stake among multiple identities, it would remain the same as if the attacker holds all the stake in one identity.

Race attack is usually considered as an implementation issue and relies on the existence of probabilistic finality. In blockchains where finality is not guaranteed, a certain number of confirmations are required before a transaction is considered complete. Typically, a recommended number is established within the network, but in some cases, a merchant or user may use a more conservative number at their discretion. Transactions handling a considerable amount of funds might require a greater number of confirmations before being accepted as valid. It is possible that a service is not configured correctly and does not wait for the recommended number of confirmations for that specific blockchain. In that case, a double spending attack is possible. An attacker could create a fork of the blockchain that meets the minimal requirements of a service, obtain the goods or benefits of that "weakly"-verified transaction, and then redirect their computing power back to the main branch of the blockchain. This way, the "weakly"-verified transaction will be soon lost in a branch of the blockchain and will not be valid for any node following the main chain. This 51% attacks are a threat to any consensus protocol.
In a PoW blockchain system, the entity with the most hashing power at a certain point in time can control the entire blockchain by creating a fork and mining on their own branch. Over time, this branch will overtake the original chain and become the new main chain. In those systems, there is a probability of multiple branches being created, but as the percentage of hashing power controlled by an entity increases past 51%, the likelihood of conflicting branches diminishes. However, even if the adversary controls the main chain, there may still be instances of branches being reversed. Additionally, miners often work together by joining mining pools to increase their chances of generating blocks

and receiving rewards. Mining pools allow nodes with low computing power to receive rewards from block rewards, but they can also inadvertently lead to 51% attacks. When a single mining pool reaches a high level of computational power, the miner community may shift to other pools in order to stabilise the network and prevent accidental 51% attacks.
In PoS protocols this attack is still viable but with a slightly different impact. In a Byzantine Fault Tolerant proof-of-stake blockchain, it is possible for a single validator or group of validators to control more than 34% of the total stake on the blockchain. In this case, a majority attack can impact the blockchain by performing finality reversion, where an already finalised block is being challenged by finalising another competing block, causing Liveness Denial or Censorship attack. Currently, there is no concrete solution to this problem, but it is mitigated by BFT-based Proof of Stake protocols with the use of absolute finality. When a 51% attack occurs, it will usually be notified by the nodes, and a community-driven fork will take place to re-establish the honest chain as the main chain.

For proof of stake protocols, however, the most effective attacks are the so-called long-range attacks. If successful, these attacks can lead to the takeover of the main chain and could result in partial or complete rewriting of the blockchain's transaction history. A long-range attack is an attack scenario where the adversary goes back to the genesis block and forks the blockchain. The attacker creates a new branch of the blockchain that has a different transaction history than the main chain. This attack is successful when the attacker's branch becomes longer than the main chain and overtakes it, replacing the original history. Long-Range attacks can be categorised into three types: *Simple*, *Posterior Corruption*, and *Stake bleeding*. These attacks are similar to selfish mining attacks in PoW protocols, as the attacker in both cases is adding blocks that they keep hidden. However, selfish mining attacks in PoW protocols cannot go back to the genesis block due to the high computational effort required, thus limiting their impact. Both attacks involve creating a fork of the main chain, where the attacker attempts to add forged blocks with potentially altered transactions. We are not going into details of this kind of attacks because they will be not be covered in the next chapters.

Chapter 3

Model Checking and PRISM

Model checking is an automatic technique for verifying correctness properties of a systems. It can be performed *automatically* and has the advantage of being able to analyse all possible states of the system. This makes it a useful tool for identifying potential issues that may not be detected through other methods, such as simulation or testing. A model checker is a tool that examines all possible states of a system by performing an exhaustive search of its finite state space. This method is used to verify whether a specific property or specification holds true or false for a given system model. It allows for determining whether a system model satisfies a particular property or not. If a system does not meet the desired properties, a model checker will produce a counterexample that shows the incorrect behaviour. This faulty trace can help to identify the cause of the failure and provide clues for resolving the issue. With enough resources, the model checker will always complete its execution and give a results in terms of yes/no. Moreover, it can be implemented by algorithms with reasonable efficiency. Model checking is typically used to check for properties that are qualitative in nature, such as: *Is the generated result OK?*, *Can the system reach a deadlock situation?*, etc. Therefore, for model checking to be effective, the properties being examined must be clearly and specifically defined.

This chapter presents Continuous Time Markov chains in Section 3.1, and, since model checking suffers from the state-space explosion problem and a proposed solution in using statistical model checking, this is discussed in Section 3.2. Section 3.3 gives an

overview of the model checker PRISM.

3.1 Continous Time Markov Chains

A continuous-time Markov Chain (CTMC) is a stochastic process in which the conditional probability of the future state at time $t + s$ is given the present state at s and all past states depends only on the present state and is independent of the past. In particular, when dealing with CTMC, we assume that transitions between states are measured by rates that correspond to separate exponential distributions that are independent, rather than independent probabilities. Formally:

Definition 3.1. *A* continuous-time Markov chain *(CTMC) is defined by a set of states S and a transition rate matrix $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$ where $\mathbf{R}(s, s')$ is the rate of making a transition from state s to s'.*

The interpretation is that the probability of moving from s to s' within $t \geq 0$ time units is $1 - e^{-\mathbf{R}(s,s') \cdot t}$. Thus, the transition rate matrix $\mathbf{R}$ assigns rates to each pair of states in the CTMC, which are used as parameters of the exponential distribution.

Example 3.1. A valid rate matrix for a CTMC could be the following

$$\mathbf{R} = \begin{bmatrix} 0 & 3 & 1 \\ 0 & 0 & 2 \\ 1 & 1 & 0 \end{bmatrix}$$

with the associated CTMC shown in Figure 3.1.

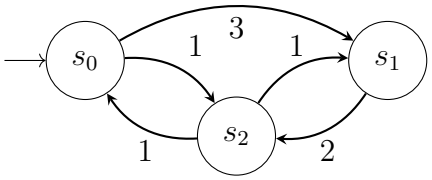


Figure 3.1: A three state CTMC.

Typically, in a state s, there is more than one state s' for which $\mathbf{R}(s, s') > 0$. This is known as a *race condition*. The first transition to be triggered determines the next state of the CTMC. The time spent in state s, before any such transition occurs, is:

$$E(s) = \sum_{s' \in S} \mathbf{R}(s, s')$$

$E(s)$ is known as the *exit rate* or *sojourn time* of state s and represents the mean time the system will be in the state s. The sojourn time is exponentially distributed with rate $E(s)$ and, thus, the higher the rate $E(s)$, the shorter the average residence time in s. It is also possible to define the average sojourn time of state s as $\frac{1}{E(s)}$. We can determine the actual probability of each state s' being the next state to which a transition is made from state s independent of the time at which this occurs; this probability is called embedded probability.

Definition 3.2. *Given a CTMC with transition rate matrix* $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$ *and* $\forall s \in S$*, let* $E(s)$ *be the sojourn time, it is possible to define the* embedded probability *as follows*

$$\mathbf{P}^{emb(C)}(s, s') = \begin{cases} \frac{\mathbf{R}(s,s')}{E(s)} & \text{if } E(s) \neq 0 \\ 1 & \text{if } E(s) = 0 \text{ and } s = s' \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Moreover, we can define the probability to move from a state s *to a particular state* s' *within* t *time units. In particular, we have:*

$$\mathbf{P}^{emb(C)}(s, s', t) = \frac{\mathbf{R}(s, s')}{E(s)}(1 - e^{E(s) \cdot t})$$

We also define the infinitesimal matrix:

$$\mathbf{Q}(s, s') = \begin{cases} \mathbf{R}(s, s') & \text{if } s \neq s' \\ -\sum_{s'' \neq s'} \mathbf{R}(s, s'') & \text{otherwise} \end{cases}$$

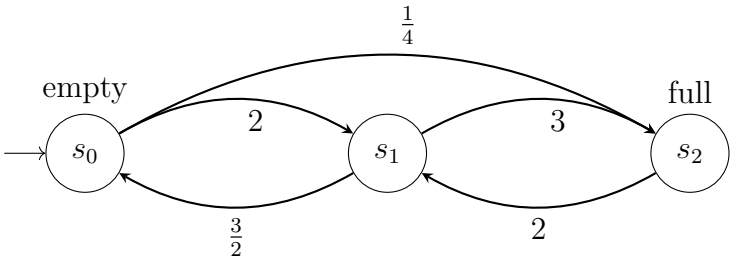


Figure 3.2: The three state CTMC C_1.

Example 3.2. Consider the following CTMC: Figure 3.2 shows a simple example of a CTMC C_1, with three states s_0 (which is the initial state), s_1 and s_2. The transitions are labelled with rates. The sojourn times are:

$$E(s_0) = \frac{9}{4}, \quad E(s_1) = \frac{9}{2}, \quad E(s_2) = 2.$$

The matrices are the following:

$$\mathbf{R} = \begin{bmatrix} 0 & 2 & \frac{1}{4} \\ \frac{3}{2} & 0 & 3 \\ 0 & 2 & 0 \end{bmatrix}, \ \mathbf{P}^{emb(C)} = \begin{bmatrix} 0 & \frac{8}{9} & \frac{1}{9} \\ \frac{1}{3} & 0 & \frac{2}{3} \\ 0 & 2 & 0 \end{bmatrix}, \ \mathbf{Q} = \begin{bmatrix} -\frac{9}{4} & 2 & \frac{1}{4} \\ \frac{3}{2} & -\frac{9}{2} & 3 \\ 0 & 2 & -2 \end{bmatrix}$$

An *infinite path* of a CTMC is a sequence $s_0 t_0 s_1 t_1 ..$ where $\mathbf{R}(s_i, s_{i+1}) > 0$ and $t_i \in \mathbb{R}_{\geq 0}$ for all i. Equivalently, a *finite path* is a sequence $s_0 t_0 \dots s_{k-1} t_{k-1} s_k$. For both cases, the time t_i represents the amount of time spent in each state s_i. For an infinite path σ, $\sigma[i] = s_i$ represents the $(i+1)$-th state of σ and $\delta(\sigma, i) = t_i$ defines the time spent in s_i. Note that for a finite state it is only defined for $i < k$.
In order to define the probabilities, it is necessary to give some notations. We call $\omega(i)$ the i-th state of a path (s_i) and $\texttt{path}(s)$ the set of all paths of the CTMC C starting from the state s. We also define $\omega @^{C} t$ as the state occupied at the time t.

Definition 3.3. *Let $s_0, s_1, \dots, s_n$ be a sequence of states where $\mathbf{R}(s_i, s_{i+1}) > 0$ for $i < n$ and $I_0, I_1, \dots, I_{n-1}$ be a sequence of non empty intervals of $\mathbb{R}_{\leq 0}$. Then, we can define a*

cylinder set $C(s_0, I_0, \ldots, I_{n-1}, s_n)$ *as the set consisting of all paths* $\sigma \in \underset{C}{\texttt{path}}(s_0)$ *such that, for* $i \leq k$*:*

$$\sigma[i] = s_i \quad \textit{and} \quad \delta(\sigma, i) \in I_i$$

Consider now $\Sigma_{\underset{C}{\texttt{path}(S)}}$, the smallest $\sigma-$algebra on $\underset{C}{\texttt{path}}(S)$ which contains all the cylinder sets $C(s_0, I_0, \ldots, I_{n-1}, s_n)$ where $s_0, s_1, \ldots, s_n$ range over all sequences of states with $s_0 = s$ and $\mathbf{R}(s_i, s_{i+1}) > 0$ for $0 \leq i < n$ and $I_0, \ldots, I_{n-1}$ range over all sequences of non-empty intervals in $\mathbf{R}_{\geq 0}$.
The probability over $\underset{C}{\texttt{path}}(S)$ can be defined inductively (Pr_s on $\Sigma_{\underset{C}{\texttt{path}(S)}}$) as the unique measure such that

- $Pr_s(C(s)) = 1$ and
- for any cylinder $C(s, I, \ldots, I_{n-1}, s_n, I', s')$, $Pr_s(C(s, I, \ldots, I_{n-1}, s_n, I', s'))$ is

$$Pr_s(C(s_0, I_0, \ldots, I_{n-1}, s_n)) \cdot \mathbf{P}^{emb(C)}(s_n, s')(e^{-E(s_n)\inf I'} - e^{-E(s_n)\sup I'})$$

We present an example to better understand the above definitions, for further details about Continuous Time Markov Chains see [9].

Example 3.3. Consider the CTMC C_1 from Figure 3.2 and the sequence s_0, I_0, s_1 with $I_0 = [0, 1]$. Using the probability measure Pr_{s_0} over $(\underset{C_1}{\texttt{path}}(s_0), \Sigma_{\underset{C}{\texttt{path}(s_0)}})$ for the cylinder set $C(s_0, [0, 2], s_1)$, we obtain:

$$\begin{aligned} Pr_{s_0}(C(s_0, [0, 2], s_1)) &= Pr_{s_0}(C_{s_0}) \cdot \mathbf{P}_1^{emb(C_1)}(s_0, s_1) \cdot (e^{-E(s_0)\cdot 0} - e^{-E(s_0)\cdot 2}) \\ &= 1 \cdot \tfrac{8}{9} \cdot (e^{-\frac{9}{4}\cdot 0} - e^{-\frac{9}{4}\cdot 2}) \\ &= \tfrac{8}{9} \cdot (1 - e^{-\frac{9}{2}}) \approx 0.879 \end{aligned}$$

Intuitively, this means that the probability of leaving the initial state s_0 and moving to state s_1 within the first 2 time units of operation is 0.879.

3.1.1 Continuous Stochastic Logic (CSL)

Continuous Stochastic Logic (CSL) was originally developed in [7] and later extended in [9]. It is based on the temporal logics CTL [25] and PCTL. It provides means to express properties on CTMCs that refer to *steady-state* and *transient behaviours*, i.e. the probability of being in each state of the chain at a particular instant in time or in the long-run, respectively. The syntax of CSL is [8]:

$$\begin{array}{rcl} \Phi & ::= & \texttt{true} \quad | \quad a \quad | \quad \neg\Phi \quad | \quad \Phi \wedge \Phi \quad | \quad \mathbf{P}_{\sim p}\left[\phi\right] \\ \phi & ::= & \texttt{X}\ \Phi \quad | \quad \Phi\ \texttt{U}^{\leq I}\Phi \end{array} \tag{3.2}$$

where a is an atomic proposition, $\sim \in \{<, \leq, >, \geq\}$, $p \in [0,1]$ and I is an interval of $\mathbb{R}_{\geq 0}$. CSL formulae are interpreted over the states of a CTMC. We say that a state $s \in S$ satisfies a CSL formula if Φ ($s \models \Phi$), if the formula is true for s.

The operators $\neg$ and $\wedge$ are the basic logical operators of negation and conjunction, respectively. From them it is possible to derive other basic logical operators:

$$\begin{array}{ll} \text{false:} & \texttt{false} \equiv \neg\texttt{true} \\ \text{disjunction:} & \texttt{false} \equiv \neg\texttt{true} \\ \text{implication:} & \phi_1 \Rightarrow \phi_2 \equiv \neg\phi_1 \wedge \phi_2 \end{array}$$

The operator $\mathbf{P}_{\sim p}\left[\phi\right]$ indicates that the probability of the path formula ϕ being satisfied from a given state meets the bound $\sim p$. Moreover, the operator $\texttt{X}\ \Phi$ represents the "next" operator, which requires the property Φ to be satisfied in the *next* step. The last operator is $\texttt{U}^{\leq I}$ which is the *time-bounded until* operator. Given the path formula $\Phi_1 \texttt{U}^{\leq I} \Phi_2$, the property holds if Φ_1 is satisfied at some time instant in the interval k and Φ_2 holds at all preceding time instants, where I is an interval of $\mathbb{R}_{\geq 0}$.

From the CSL syntax presented in (3.2), it is also possible to define the *eventually* operator $\texttt{F}$ as an operator indicating that the property ϕ is eventually true. Formally:

$$\texttt{F}\phi \equiv \texttt{true}\ \texttt{U}\ \phi$$

The time-bounded version of the eventually operator is $\mathtt{F}^{\leq I}$ and states that the property ϕ becomes true in the interval I:

$$\phi \equiv \mathtt{true}\ \mathtt{U}^{\leq I}\ \phi$$

Examples of usage of these properties will be detailed in Section 3.3.

3.1.2 CSL Model Checking

We consider a model checking algorithm for CSL over CTMCs. The inputs to the algorithm are a labelled CTMC C and a CSL formula Φ. The output is a set of states $States(\Phi) = \{s \in S \mid s \models \Phi\}$. The algorithm first constructs the parse tree of the formula Φ and, working upwards towards the root of the tree, it recursively computes the set of states satisfying each subformula. At the end, the algorithm determines whether each state in the model satisfies Φ or not. For the operator $\mathbf{P}_{\sim p}[\cdot]$ the model checking is not straightforward and it will be briefly covered below by analysing the different settings, for further details see [9].

$\mathbf{P}_{\sim p}[\mathtt{X}\Phi]$. In this situation, the probability is determined only by the probability of transitioning to the next immediate state. As a result, it can be verified using PCTL algorithms by incorporating the embedded probability $\mathbf{P}^{emb(C)}$. This requires the probabilities of the immediate transitions from s. We define the vector $\underline{Prob}(\mathtt{X}\Phi)$ as follows:

$$\underline{Prob}(\mathtt{X}\Phi) = \mathbf{P}^{emb(C)} \cdot \underline{\Phi}$$

where $\underline{\Psi}$ is a state-indexed column vector defined as

$$\underline{\Phi}(s) = \begin{cases} 1 & \text{if } s \in States(\Phi) \\ 0 & \text{otherwise} \end{cases}$$

Example 3.4. Considering the CTMC in Figure 3.2, and the CSL formula $\mathbf{P}_{\geq 0.5}[\mathtt{X}\ full]$.

The column vector is $\underline{\Phi}(s) = (0,0,1)^T$ because only in s_2 the property Φ (*full*) is verified. Thus, we calculate the probability in the following way:

$$\begin{aligned} \underline{Prob}^{C_1}(\mathtt{X}\Phi) &= \mathbf{P}^{emb(C_1)} \cdot \underline{\Phi} = \\ &= \begin{bmatrix} 0 & \frac{8}{9} & \frac{1}{9} \\ \frac{1}{3} & 0 & \frac{2}{3} \\ 0 & 2 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{9} & \frac{2}{3} & 0 \end{bmatrix} \end{aligned} \tag{3.3}$$

Thus, since the probability is not greater or equal of 0.5 for any state of the CTMC, we can conlude that the formula is not true.

$\mathbf{P}_{\sim p}\left[\Phi_1 \, \mathtt{U}^{\leq I} \, \Phi_2\right]$ For this operator, we need to determine the probabilities $Prob^C(s, \Phi_1 \, \mathtt{U} \Phi_2)$ for all the states s and with I an interval. It is important to differentiate between three distinct scenarios, but we only analyse the case in which $I = [0,t]$ with $t \in \mathbb{R}_{\geq 0}$. In this scenario, calculating the probabilities involves finding the minimal solution for the following set of integral equations:

$$\begin{aligned} &Prob^C(s, \Phi_1 \mathtt{U}^{[0,t]} \Phi_2) = 1 \quad \text{if } s \in States(\Phi_2) \\ &Prob^C(s, \Phi_1 \mathtt{U}^{[0,t]} \Phi_2) = 0 \quad \text{if } s \in States(\neg\Phi_1 \wedge \neg\Phi_2) \\ &Prob^C(s, \Phi_1 \mathtt{U}^{[0,t]} \Phi_2) = \\ &\quad = \int_0^t \textstyle\sum_{s' \in S} \mathbf{P}^{emb(C)}(s,s') \cdot E(s) \cdot e^{-E(s) \cdot x} \cdot Prob^C(s', \Phi_1 \mathtt{U}^{[0,t-x]} \Phi_2) dx \text{ otherwise} \end{aligned}$$

We now define $C[\Phi] = (S, s, \mathbf{R}[\Phi], L)$ where $\mathbf{R}[\Phi] = R(s,s')$ if $s \models \Phi$ and 0 otherwise (L represents the set of the lables of the CTMC C). Hence, we have that:

$$Prob^C(s, \Phi_1 \mathtt{U}^{[0,t]} \Phi_2) = \sum_{s' \models S} \pi_{s,t}^{C[\neg\Phi_1 \vee \Phi_2]}(s')$$

where $\pi_{s,t}^C$ is the *transient probability* defined as $\pi_{s,t}^C = Pr_s\{\omega \in \underset{C}{\mathtt{path}}(s) \mid \omega @ t\}$. Since a path in a CTMC cannot exit a state satisfying Φ_2 once it reaches one, and will never be able to reach a state satisfying Φ_2 if it enters one satisfying $\neg\Phi_1 \wedge \neg\Phi_2$, the probability of the path formula $\Phi_1 \, \mathtt{U}^{[0,t]} \Phi_2$ being satisfied in CTMC C is equivalent to the transient

probability of being in a state satisfying Φ_1 at time t in CTMC $C[\neg\Phi_1 \vee \Phi_2]$. Then, we have:

$$Prob^C(s, \Phi_1 \mathrm{U}^{[0,t]} \Phi_2) = \sum_{i=0}^{+\infty} (\gamma_{i,q\cdot t} \cdot (\mathbf{P}^{\mathrm{uniff}(C[\neg\Phi_1 \vee \Phi_2])})^i) \cdot \underline{\Phi}_2)$$

where $\mathbf{P}^{\mathrm{uniff}(C)} = I + \frac{\mathbf{Q}}{q}$ is called *uniformised matrix* and $q \geq \max\{E(s) \mid s \in S\}$ is the uniformisation rate.

3.2 Statistical Model Checking

Model checking suffers from the state-space explosion problem. This is particularly true when the system is composed of multiple subsystems or dynamic data types are used. An alternative method to avoid the need to explore the entirety of the state-space of a model is to use statistical model checking, which has been proposed as a solution [76].

The basic concept of statistical model checking is to simulate the system, observe it, and then use statistical analysis to determine if the system satisfies a property to a certain level of confidence. This approach balances between testing and traditional model checking techniques. Simulation-based methods are less demanding in terms of memory and time compared to exhaustive methods and are often the only viable option. Stochastic model checking only requires that the system to be analysed can be simulated using discrete-event simulation and that there is a probability space on executions of the system and is directly applicable to systems whose behaviour can modelled by a Discrete Time Markov Chain (DTMC) or Continuous Time Markov Chain (CTMC) [84].

Formally, statistical model checking is a method that uses simulation-based techniques to analyse a stochastic system (S, T) and determine if it satisfies a specific property (ϕ). It can be used to answer two types of questions:

(a) **Qualitative:** Is the probability that (S, T) satisfies ϕ greater or equal to a certain threshold?

(b) **Quantitative:** What is the probability that (S, T) satisfies ϕ?

The quantitative question can be answered by using probability estimation techniques that are based on the Chernoff-Hoeffding bound [24, 47]. It mainly involves calculating

an estimate probability of the actual probability that the system S satisfies the property (ϕ). The qualitative question is mainly addressed through a hypothesis testing approach, which will not be discussed since in our work we focus only on quantitative properties.

The core idea is to estimate the probability one wants to calculate using a statistical estimator based on a limited number of events from the random space, in particular, a number N of runs (or paths) of the model. One benefit of this approach is that we can obtain these paths directly from the model without the need to construct the graph. This step is referred to as *generation*. Once the paths have been obtained, the probability is estimated by testing the property on each of them – this is the *verification* step. In particular, we compute $\frac{n}{N}$, where n is the number of paths which validate the property. The estimator we use asymptotically approaches the true probability, so using a limited number of runs does not provide a guarantee of the result. It is important to understand the reliability of the experiment for a certain number of runs. This reliability characterises three different methods to perform statistical model checking for quantitative properties:

- CI (Confidence Interval)
- ACI (Asymptotic Confidence Interval)
- APMC (Approximate Probabilistic Model Checking)

Since in our analyses we will apply the CI methods, the other two approaches will not be detailed.

Confidence Interval (CI). A confidence interval (CI) is an interval of a certain width $2w$, i.e. an interval of the form $[Y - w, Y + w]$, which is estimated such that, when the estimation is repeated several times, the true probability (say X) of the query $\mathbf{P}_{=?}[\phi]$ falls within this interval $100 \times (1 - \alpha)\%$ of the time, where α is the level of confidence. In particular, the width of the interval w can be determined for a given α and number of paths N by using the formula:

$$w = q \times \sqrt{\frac{v}{N}}$$

where q is a quantile (for a probability of $1-\alpha/2$) from the Student's t-distribution with $N-1$ degrees of freedom, and v is an estimation of the variance for X. Similarly, the number of iterations required can be determined from w and α as:

$$N = \frac{(v \times q^2)}{w^2}$$

where q and v are as previously defined.

3.3 The Model Checker **PRISM**

PRISM [52] is a probabilistic model checker that, given a formal description of a system, called the *model*, computes the likelihood of the occurrence of certain events. The model checker supports different kinds of probabilistic formalism which give semantics to the model. We refer to [51] for a full account of PRISM.

A PRISM system is a parallel composition of interacting modules, where each module represents a (sequential) agent/process. The internal state of a module is determined by the values assigned to its variables, whereas the overall state of a system is determined by the internal states of all its modules. The behaviour of a module is defined by a set of commands that specify how and under which conditions the module performs a transition and updates its internal state. The syntax of a command is

`[a] guard -> rho`$_1$`:update`$_1$`+...+rho`$_n$`:update`$_n$`;`

where a is called action and may be omitted; guard is a predicate over the state variables of the module and those of other modules; update$_i$ defines the changes of module's internal state, i.e., a list of assignments to its state variables; and rho$_i$ is the rate at which update$_i$ is executed. The meaning of the above command is the following: when guard is true, the module chooses a transition (the operator + denotes a choice) according to the rate rho$_i$ associated to that update.

PRISM semantics constrains modules retaining actions with the same name to *synchronise* with the corresponding commands, *i.e.* the modules execute the commands with

action a at the same time. The example in Listing 3.1, taken from the documentation,[1] helps understand the semantics of modules. It models an N-place queue of jobs and a server that removes jobs from the queue and processes them.

```
1  const int N = 10;
2  const double mu = 1/10;
3  const double lambda = 1/2;
4  const double gamma = 1/3;
5
6  module queue
7       q : [0..N];
8
9       [] q<N -> mu:(q'=q+1);
10      [] q=N -> mu:(q'=q);
11      [serve] q>0 -> lambda:(q'=q-1);
12 endmodule
13
14 module server
15      s : [0..1];
16
17      [serve] s=0 -> 1:(s'=1);
18      [] s=1 -> gamma:(s'=0);
19 endmodule
```

Listing 3.1: A PRISM specification modelling a N-place queue and a server.

The modules queue (line 6 to 12) and server (line 14 to 19) synchronise on the action serve. Module queue has an integer variable q (defined in line 7) representing its size (the constant N defines the capacity of the queue). Transitions describe the operations on the queue. The first one (line 9) inserts a new element with rate mu, if the queue is not full (q<N); the insertion is modelled by increasing the value of q. Note that PRISM uses the prime notation to denote the new value of a variable, in our case q' = q + 1. The second command (line 10) says that no new element is inserted when the queue is

[1] http://www.prismmodelchecker.org/manual/ThePRISMLanguage/Example2

full. The last command (line 11) removes an element, and it is performed provided that the server can consume an element; for this reason it has the action name `serve` that constrains `server` to perform a transition with the same name. In the module `server`, the boolean variable `s` (line 15) defines whether the server is busy or not. When it is idle, the command at line 17 allows the server to synchronise with `queue` through the action `serve`. After this synchronisation the server updates its state to busy (`s'=1`). The rate of the synchronisation is the product of the two individual rates (in this case, `lambda`×1). The second command (line 18) of `server` states that a busy server (`s=1`) may complete its task with rate `gamma`.

As already explained, in this thesis we focus on Continuous Time Markov Chains (CTMC) models that are transition systems (as the one above) where each transition is labeled by a positive real number, called *rate*. In particular, we are interested in *the probability of reaching states with a given property within a certain time*. A basic property of Markov chains is that the events are independent from the previous events in the history – the Markov property. Therefore, the above probability is a function of the product of the probabilities in the intermediate states (this function is not simple because one has to consider all the possible partitions of t and all the possible paths to reach the state from the initial step).

In the `PRISM` framework, properties of CTMC models are expressed in CSL. In particular, the formulas we use in this paper have always the form

P=?[**F**<=T property]

and express the probability that `property` is *eventually* true in a state of the model within t time units (starting from the initial state). In particular, *eventuality* is expressed by the operator `F` of CSL logic; the operator `?` asks the model checker to produce a numeric value for the probability. For example, consider the following code that implement a two-items queue

```
1  module C
2      x : [0..2] init 0;
3
4      [] x=0 -> 2 : (x' = 1);
5      [] x=1 -> 3 : (x' = 2);
```

```
6       [] x=1 -> 5 : (x′ = 0);
7       [] x=2 -> 2 : (x′ = 1);
8
9   endmodule
10
11  label "full" = x = 2;
12  label "empty" = x = 0;
```

where the variable x can assume three possible values: 0 (the queue is empty), 1 (the queue has one element) and 2 (the queue is full, *i.e.* it has two elements). According to lines 5 and 6, from state x=1 the system can evolve either in x=2 with rate 3 and in x=1 with rate 5. The probability that the two-items queue is "full" within 10 time units is denoted by the CSL formula

```
P=?[F<=10 "full"].
```

To actually compute this probability, PRISM performs statistical model checking.

Part II

Contributions

Chapter 4

PRISM+

The formal model for defining the blockchain protocols is an extension of PRISM [52]. PRISM has been chosen for two reasons. First, because it is a simple process calculus with formal stochastic semantics that uses rates of actions as parameters of an exponential distribution, which is a standard feature of Bitcoin actions of mining and broadcasting [12, 85, 33, 17]. Secondly, because PRISM has a tool for analysing stochastic systems that can be used for complementing our theoretical results with practical simulations.

However, as it is, PRISM falls short to model faithfully blockchain protocols because it misses the datatypes of blocks and ledgers. Therefore, following the description in [62], we have defined the values of blocks, sets, maps, and ledgers and the corresponding operations.

This chapter presents the modelling language and our extension of PRISM, PRISM+.

4.1 The Modelling Language

To define PRISM+ we use a set of *action names* A, ranged over a, b, ..., a set of *module names* ranged over $\mathtt{M}$, $\mathtt{M}_1$, ..., and a set of *variables*, ranged over by x, y, z. Let α ranges over $A \cup \{\varepsilon\}$, where ε indicates *no-action*; let also ρ range over reals (called `double`).

A PRISM+ program P is a parallel composition of *modules*, that is

$$\mathtt{P} \;=\; \mathtt{M}_1 \;\mid\; \ldots \;\mid\; \mathtt{M}_n$$

where M | M′ is the parallel composition of modules M and M′ synchronising only on actions appearing in both M and M′. Let *actions*(M) be the set of actions in A that occur in M. A module M is defined by the syntax

$$\begin{array}{rcl}
\mathtt{M} & ::= & \mathtt{module}\ \mathtt{M}\mathtt{:}\ \mathtt{D}\ \mathtt{C}\ \mathtt{endmodule} \\
\mathtt{D} & ::= & \mathtt{T}\ x = v\mathtt{;} \;\mid\; \mathtt{T}\ x = v\mathtt{;}\mathtt{D} \\
\mathtt{T} & ::= & \mathtt{integer} \mid \mathtt{double} \mid \mathtt{bool} \mid \mathtt{block} \mid \mathtt{ledger} \mid \mathtt{set} \\
 & & \mid \mathtt{map}
\end{array}$$

That is, a module has a name, a sequence D of *local variable declarations* with initialisations and a *set of commands* C. It is assumed that pairwise different modules in a PRISM+ program have different names and have also different local variables names.

Sets of commands C are written $c_1\mathtt{;}\ \ldots\ \mathtt{;}\, c_m$, where every c has the form:

$$\begin{array}{rcl}
c & ::= & [\alpha]\, e \to \sum_{i\in I} \rho_i : \mathtt{update}_i \\
\mathtt{upd} & ::= & \varepsilon \;\mid\; x' = e\ \mathtt{\&}\ \mathtt{upd} \\
e & ::= & v \;\mid\; x \;\mid\; e\ \mathtt{op}\ e \;\mid\; !e \\
v & ::= & \mathit{true} \mid \mathit{false} \mid \mathit{integers} \mid \mathit{doubles} \mid \mathit{ledgers} \mid \mathit{set} \\
 & & \mid \mathit{blocks} \mid \mathit{maps} \\
\mathtt{op} & ::= & - \mid + \mid * \mid = \mid \neq \mid \mathtt{\&}
\end{array}$$

In a command $[\alpha]\, e \to \sum_{i\in I} \rho_i : \mathtt{upd}_i$, α may be either empty or an action, e, called *guard*, is a boolean expression over all the variables in the program (including those belonging to other modules), and the right hand-side of the arrow describes a *transition*. In particular, when α is empty, if e is true then one of the corresponding updates may be performed. Each update is defined by giving new values of *the variables in the module*, possibly as a function of other variables. Each update has also a rate, which will be given to the corresponding transition. Updates are written with the prime symbol:

$x' = e$ means that, if v is the value of e in the current state then the value of x in the *next state* is v. We assume that, in an update $x_1{}' = e_1 \ \& \ \ldots \ \& \ x_n{}' = e_n$, left hand-side variables are all different.

When α is an action then the transition must be performed simultaneously with the other modules in parallel that have the same action (i.e. the modules *synchronise*). This is the standard CSP parallel composition [46]. The rate of the overall transition is equal to the product of the individual rates. Since the product of rates does not always meaningfully represent the rate of a synchronised transition, PRISM uses the technique to make exactly one action *active*, with a generic rate, and all the others *passive*, with rate 1. The rate of a synchronisation is therefore defined by the unique active action.

Semantics. The semantics of a PRISM+ program is defined as a transition system whose states s are maps $[x_1 \mapsto v_1, \ldots, x_n \mapsto v_n]$ where $\{x_1, \ldots, x_n\}$ is the set of local variables of the program's modules. The transition relation uses the following auxiliary definitions:

- $s[x \mapsto v]$ is the state

$$(s[x \mapsto v])(y) \stackrel{\text{def}}{=} \begin{cases} v & \text{if } y = x \\ s(y) & \text{otherwise} \end{cases}$$

- $[\![e]\!](s)$ returns the value of an expression e in the state s. The value is computed by replacing the variables with their values in s and evaluating the operations. The formal definition is omitted because standard.

- $[\![\texttt{upd}]\!](s)$ returns the state s' defined as follows:

$$[\![x_1{}' = e_1 \ \& \ \ldots \ \& \ x_n{}' = e_n]\!](s) \stackrel{\text{def}}{=} s[x_1 \mapsto [\![e_1]\!](s), \ldots, x_n \mapsto [\![e_n]\!](s)]$$

The transition relation of PRISM+ is defined in Table 4.1 where we let $\mathscr{M}$ range over parallel compositions of modules and we assume $|$ to be commutative. We use the judgment $\mathscr{P} \Vdash s \xrightarrow{\alpha,\rho} s'$ meaning that the program $\mathscr{P}$ transits from s to s' with an action α and rate ρ. The auxiliary judgment $\mathscr{M} \Vdash s \xrightarrow{\alpha,\rho} \texttt{upd}$ collects all the updates in the

synchronising modules in $\mathscr{M}$ (according to our assumptions, different updates modify different variables). Rule [Upd] defines the semantics of a command. We write $c \in \mathtt{M}$ if

$$\text{[Upd]}\quad \frac{[\alpha]\, e \to \sum_{i \in I} \rho_i : \mathtt{upd}_i \in \mathtt{M} \qquad [\![e]\!](s) = \mathtt{true}}{\mathtt{M} \Vdash s \xrightarrow{\alpha,\rho_i} \mathtt{upd}_i}$$

$$\text{[Sync]}\quad \frac{\mathscr{M} \Vdash s \xrightarrow{a,\rho} \mathtt{upd} \qquad \mathscr{M}' \Vdash s \xrightarrow{a,\rho'} \mathtt{upd}'}{\mathscr{M} \mid \mathscr{M}' \Vdash s \xrightarrow{a,\rho\times\rho'} \mathtt{upd} \,\&\, \mathtt{upd}'} \qquad \text{[Nosync]}\quad \frac{\mathscr{M} \Vdash s \xrightarrow{\alpha,\rho} \mathtt{upd} \qquad \alpha \notin \mathit{actions}(\mathtt{M})}{\mathscr{M} \mid \mathtt{M} \Vdash s \xrightarrow{\alpha,\rho} \mathtt{upd}}$$

$$\text{[Program]}\quad \frac{\mathtt{M}_1 \mid \cdots \mid \mathtt{M}_n \Vdash s \xrightarrow{\alpha,\rho} \mathtt{upd} \qquad \mathscr{P} = \mathtt{M}_1 \mid \cdots \mid \mathtt{M}_n}{\mathscr{P} \Vdash s \xrightarrow{\alpha,\rho} [\![\mathtt{upd}]\!](s)}$$

Table 4.1: The semantics of the PRISM+ language.

$\mathtt{M} = \mathtt{module}\ \mathtt{M}\ \mathtt{:}\ \mathtt{D}\ \mathtt{C}\ \mathtt{endmodule}$ and $c \in \mathtt{C}$. If e is true, then an update $\mathtt{upd}_i$ is enabled with rate ρ_i and label α. The update $\mathtt{upd}_i$ is a set of evaluated variables expressed as a conjunction of assignments. Rule [Sync] collects commands of synchronising modules. We notice that the rate is the product of the rates of every single transition, which is actually the one of the unique active transition. Rule [Nosync] enables the interleaving of transitions (because of commutativity of $\mid$, it also covers the symmetric rule). A PRISM+ program is a parallel composition of modules; its semantics is described in [Program].

4.2 Our Extension

PRISM+ extends PRISM by adding a native support for expressing and manipulating data types such as lists and trees and data types specifically designed for modelling blockchain protocols such as block and ledger. The goal of our extension is to provide a generic set of primitives that can be used to model and analyse different kinds of blockchain

protocols. This section describes the data types and the operations of PRISM+ and their prototype implementation.

4.2.1 The Data Types

PRISM has not been designed to be extended with plugins. Therefore, to implement our data types and operations we had to modify the PRISM source code and provide a new software package that includes our extensions[1,2].

In particular, we have extended the syntax of the PRISM model checker by upgrading the parser (the file PrismParser.jj). The supported expression in PRISM are literal values (12, 3.141592, true, false, etc.), identifiers (corresponding to variables, constants, etc.) and some operators. Thus, the extension of PRISM expressions with the new types has led to extending the PRISM abstract class Expression. In particular, the data types that have been implemented in PRISM+ are block, ledger, set, and map and the operations between them. In the rest of the section, we will cover the main aspects of the implementation of each data type.

Blocks

The first data type we will discuss is the block data type. In our model, we simplify the concept of a blockchain block by only requiring that it has a unique name. We do not consider the specific information items such as transactions, nonce, timestamp, Merkle root, and parent pointer, as we assume all blocks and transactions are valid. For this reason, our blocks are terms of the form $\{\mathtt{m},\mathtt{n};\rho\}$, where

- m, n, called *name*, is such that m is a creator's name and n is a unique numeric label. In particular, n is a personal counter of each node which is incremented every time the node creates a new block;

- the term ρ, called *parent*, is the name of the parent block to which $\{\mathtt{m},\mathtt{n};\rho\}$ points.

[1] https://github.com/adeleveschetti/bitcoin-analysis
[2] https://github.com/adeleveschetti/casper-analysis

For instance, {m3,0;m4,7} denotes a block named (m3,0), (which is the first block created by the node m3), and whose parent is the block named (m4,7) (which is the seventh block created by the node m4). The block genesis, called *genesis block* is the root of every ledger and is represented as {genesis,0;genesis,0}.

```
1  public class ExpressionBlock extends Expression{
2
3      private ExpressionString module = null;
4      private ExpressionLiteral value = null;
5      private ExpressionString moduleParent = null;
6      private ExpressionLiteral valueParent = null;
7      private int height = 0;
8
9      public ExpressionBlock(ExpressionString m1, ExpressionLiteral v1,
           ↪ ExpressionString m2, ExpressionLiteral v2, int h)
10     {
11         module = m1;
12         value = v1;
13         moduleParent = m2;
14         valueParent = v2;
15         type = new TypeBlock();
16         height = h;
17     }
18 }
```

Listing 4.1: The ExpressionBlock fields and constructor.

The implementation is straightforward, the field module represents the name of the node that has created this block, while the value is the counter. The fields moduleParent and valueParent represent, respectively, the name of the node that created the parent block and the node's counter. Additionally, a block in our model has a field that indicates its position or height in the ledger, height. The methods implemented for the ExpressionBlock class are the standard *getters* and *setters*.

Ledgers

As previously outlined, our abstraction of a ledger is a tree of blocks (see Section 4.2.3), with its blockchain being the sequence of blocks starting from the leaf block at the maximum depth and leading to the genesis block. To represent the whole ledger we use the auxiliary data structure `TreeList` (which will be discussed later on) and the ledger class contains the informations about the leaf block at maximal depth and, in the case of the Hybrid Casper protocol, also the last justified block and the last finalised block.

```
1  public class ExpressionBlockchain extends Expression{
2    private String name;
3    private int length;
4    private TreeList<ExpressionBlock> blocks;
5    private ExpressionBlock lastFinalized;
6    private ExpressionBlock lastJustified;
7
8    public ExpressionBlockchain(ExpressionBlock b){
9       name = null;
10      blocks = new TreeList<ExpressionBlock>();
11      blocks.addHead(b);
12      length = blocks.size();
13      lastFinalized = b;
14      lastJustified = b;
15      lastAdded = b;
16   }
17 }
```

Listing 4.2: The `ExpressionBlockchain` fields and constructor.

Since, the genesis block is the only block that all nodes are aware of at time zero, in our models a blockchain is always initialised as containing only the genesis block. Thus, the constructor reported in Listing 4.2 is used with the input parameter `b` $\equiv$ {`genesis,0; genesis,0`}. In this way, a new tree of blocks is instantiated and the last finalised block, the last justified block and the last added are all set to the genesis block, following the definitions of the literature [19, 21].

Besides the setters and the getters, the `ExpressionBlockchain` class is packed also with the method `addBlock`, shown in Listing 4.3. This method adds the block taken as parameter in the ledger, also updating the `lastAdded` field.

```
1    public ExpressionBlockchain addBlock(Expression block) {
2        this.blocks.addLeaf((ExpressionBlock) block);
3        lastAdded = this.blocks.getHead();
4        return this;
5    }
```

Listing 4.3: The `ExpressionBlockchain` method `addBlock`.

Sets

Sets are collections of blocks. In `PRISM+`, we model them as `ExpressionSet` which is data structure that can be approximated as a list of blocks without repetitions.

```
1  public class ExpressionSet extends Expression{
2
3      private String name;
4      private int length;
5      private List<ExpressionBlock> blocks;
6
7      public ExpressionSet(){
8          name = null;
9          blocks = null;
10         length = 0;
11     }
12 }
```

Listing 4.4: The `ExpressionList` class.

We ensure that there are no duplicates in the set by verifying every time a new block is added using the method `addBlock` reported in Listing 4.5. In particular, if the set is `null`, a new set is created and the block is added to it. If the set already exists, the block is checked for duplicates before being added to the set.

```
1    public ExpressionSet addBlock(Expression operand2) {
2       if(this.blocks==null) {
3          this.blocks = new ArrayList<ExpressionBlock>();
4          this.blocks.add((ExpressionBlock) operand2);
5          length = blocks.size();
6          lastAdded = (ExpressionBlock)operand2;
7       }
8       else {
9          if(!this.blocks.contains((ExpressionBlock) operand2)) {
10            this.blocks.add((ExpressionBlock) operand2);
11            lastAdded = (ExpressionBlock)operand2;
12         }
13      }
14      return this;
15   }
```

Listing 4.5: The `ExpressionList` class.

Maps

In the Hybrid Casper model, maps are utilised to keep track of the stakes of validators and the votes for each checkpoint.

```
1  public class ExpressionMap extends Expression{
2     private String name;
3     private int length;
4     private List<Pair> votedBlocks;
5     private Pair lastModified;
6     public ExpressionMap(){
7        name = null;
8        votedBlocks = null;
9        length = 0;
10       lastModified = null;
11    }}
```

Listing 4.6: The `ExpressionMap` class.

Maps are represented by means of the `ExpressionMap` data structure and are used to store votes for each block. In particular, a map is composed by a name, a length of the list, a list of pairs and the last modified pair.

```
1  public ExpressionMap addVotedBlock(Expression block, int index, int stake) {
2        boolean flag = false;
3        int j = 0;
4        if(this.votedBlocks!=null) {
5           for(int i=0; i<this.votedBlocks.size(); i++) {
6              if(this.votedBlocks.get(i).getBlock().equals((ExpressionBlock) block)) {
7                 flag = true;
8                 j = i;
9              }
10          }
11       }
12       if(!flag  this.votedBlocks==null) {
13          if(this.votedBlocks==null) {
14             this.votedBlocks = new ArrayList<Pair>();
15          }
16          Pair newPair = new Pair ();
17          newPair.setBlock((ExpressionBlock) block);
18          newPair.setVote(stake,index);
19          this.votedBlocks.add(newPair);
20          this.lastModified = newPair;
21          length = votedBlocks.size();
22       }
23       else {
24          this.votedBlocks.get(j).setVote(stake,index);
25          this.lastModified = this.votedBlocks.get(j);
26       }
27       return this;
28    }
```

Listing 4.7: The `addVotedBlock` method.

The auxiliary data type pair `(block, votes)` consist of a block and the list of votes received for that block. The method "addVotedBlock" (as shown in Listing 4.7) is used to add a block and its corresponding vote to the map. It takes in the block, the index of the voting validator and the validator's stake as parameters. The method checks if the block the validator is voting for is already in the map or not. If it is present, the new vote is added to the correct index. If it is not, a new pair with the new block is created and added to the map.

4.2.2 The Operations

After extending the data types in our model, we realised that we also needed new operations to analyse and modify our data structures. Therefore, we added new operations to the existing `PRISM` tool by extending the `ExpressionFunc` class, which defines the built-in operations. With these additions, `PRISM+` now supports a variety of operations in addition to the standard ones defined in `PRISM` (such as `min(..)`, `max(..)`, `floor(..)`, `ceil(..)`, `round(..)`, `pow(..)`, `mod(..)`). The following section lists the newly introduced operations, organised by the main data type they work on.

The operations on blocks are:

- `createB(block, blockchain, int)` creates and returns a new block as a child of the last valid block of this blockchain. The integer represents the number of blocks created by that specific node of the network and it is used to make the name of the block unique;
- `getHeight(block)` returns the height of the block taken as parameter.

The operations implemented on ledgers are:

- `canBeInserted(blockchain,block)` returns `true` if the block taken as parameter can be inserted in the ledger, `false` otherwise. We recall that a block can be inserted if the father of the block is already in the ledger;
- `addBlock(blockchain,block)` inserts the block in the blockchain and returns the updated ledger (precondition: `canBeInserted(..)=true`). The function also updates the `lastAdded` field if the block is the a leaf at maximal depth;

- `diffCheckpoints(blockchain, int)` checks if there are conflicting checkpoints at the given height in the blockchain. It is used to check the consistency of the ledgers in the Hybrid Casper protocol. To make the function as versatile as possible, an additional integer parameter is required which represents the length of the epoch in the current system.

```
1  private Object evaluateDiffChekpoints(EvaluateContext ec) throws
       ↪ PrismLangException
2      {
3          ExpressionBlockchain B = getOperand(0).evaluateBlockchain(ec);
4          int epoch = getOperand(1).evaluateInt(ec);
5          ArrayList<ExpressionBlock> blocks = B.getBlocks().getNodes();
6          int fork = 0;
7          for(int i=0; i<blocks.size(); i++) {
8              ExpressionBlock b1 = blocks.get(i);
9              for(int j=i; j<blocks.size(); j++) {
10                 ExpressionBlock b2 = blocks.get(j);
11                 if(b1.getHeight()%epoch==0&&b2.getHeight()%epoch==0 b1.
                       ↪ getHeight()==b2.getHeight()) {
12                     if(!b1.equals(b2)) {
13                         fork++;
14                     }
15                 }
16             }
17         }
18         return fork;
19     }
```

Listing 4.8: The implementation of `diffCheckpoints` operation.

The function takes all the blocks in the ledger with the operation `getNodes()` of the class `TreeList`, not considering the different paths leaf-genesis block. Thus, if two blocks are both checkpoints (their height is a multiple of the length of the epoch) and they have the same height, then there is a fork of checkpoints. The

function returns the length of this kind of fork;

- `calcFork(blockchain,..., blockchain)` returns the length of the fork between the blockchains (Listing 4.9).

```
1  private Object evaluateCalcFork(EvaluateContext ec) throws
       ↪ PrismLangException{
2        ExpressionBlockchain B0 = getOperand(0).evaluateBlockchain(ec);
3        ArrayList<ExpressionBlockchain> bchains = new ArrayList<
             ↪ ExpressionBlockchain> ();
4        bchains.add(B0);
5        int n = getNumOperands();
6        ExpressionBlockchain tmpB ;
7        for (int i = 1; i<n; i++) {
8           tmpB = getOperand(i).evaluateBlockchain(ec);
9           bchains.add(tmpB);
10       }
11       ArrayList<ArrayList<ExpressionBlock>> mainChains = new ArrayList
             ↪ <ArrayList<ExpressionBlock>> ();
12       ArrayList<ExpressionBlock> tmpChain;
13       ArrayList<Integer> lenghts = new ArrayList<Integer> ();
14       int maxLen = 0;
15       for(int i = 0; i<n; i++) {
16          tmpChain = bchains.get(i).getMainChain();
17          mainChains.add(tmpChain);
18          lenghts.add(tmpChain.size());
19          if(lenghts.get(i)>maxLen) {maxLen = lenghts.get(i);}
20       }
21       ArrayList<ArrayList<ExpressionBlock>> toCalc = new ArrayList<
             ↪ ArrayList<ExpressionBlock>> ();
22       for (int i = 0; i<n; i++) {
23          if(mainChains.get(i).size() == maxLen) {
24             toCalc.add(mainChains.get(i));
25          }
```

```
26          }
27          int diff = toCalculate(toCalc,maxLen);
28          return diff;
29      }
```

Listing 4.9: The implementation of `calcFork` operation.

The function collects all the blockchains into a single array and passes it to the `toCalculate(..)` function (shown in Listing 4.10). This function computes the length of the fork recursively by comparing the elements at height `len-1`, where `len` is decreased with each call of the function. It is worth noting that the implementation ensures that if there are at least two different blockchains, a fork exists.

```
1  private int toCalculate(ArrayList<ArrayList<ExpressionBlock>> list,
       ↪ int len) {
2          if(len==1) { return 0; }
3          else {
4              int diff = 0;
5              ExpressionBlock b = list.get(0).get(len-1);
6              for(int i=1; i<list.size(); i++) {
7                  if(!list.get(i).get(len-1).equals(b)) {
8                      diff = 1;
9                  }
10                 b = list.get(i).get(len-1);
11             }
12             return diff+toCalculate(list,len-1);
13         }
14     }
```

Listing 4.10: The implementation of `toCalculate` auxiliary function.

- `updateFin(blockchain, block)` updates the field `lastFinalised` of the corresponding blockchain with the block taken as parameter;

- `updateJust(blockchain, block)` is similar to the previous function, the only distinction is that it updates the `lastJustified` field.

For what concerns the set data type, the operations are almost standard:

- `extractCheckpoint(set,block)` returns the block taken as parameter if it is contained in the set, the genesis block otherwise. This function is used for the process of voting in the Hybrid Casper protocol;

```
1  private Object evaluateExtractCheckpoint(EvaluateContext ec) throws
       ↪ PrismLangException
2      {
3         ExpressionSet list = getOperand(0).evaluateList(ec);;
4         ExpressionBlock block = getOperand(1).evaluateBlock(ec);
5         ExpressionBlock retBlock = null ;
6         if(list.getBlocks()!=null && list.getBlocks().size()>0) {
7            if(!list.getBlocks().contains(block)) {
8               retBlock = list.getBlocks().get(0);
9            }
10           else {
11              boolean flag = false;
12              for(int i=0; i<list.getBlocks().size()-1 && !flag; i++) {
13                 if(list.getBlocks().get(i).equals(block)) {
14                    flag = true;
15                    retBlock = list.getBlocks().get(i+1);
16                 }
17              }
18           }
19        }
20        return retBlock;
21     }
```

Listing 4.11: The implementation of `extractChecpoint` operation.

- `addBlockSet(set,block)` adds the block and returns the set updated;

- `removeBlock(set,block)` delete the block from the set, the function also returns the set;
- `isEmpty(set)` returns `true` if the set is empty, `false` otherwise;
- `extractBlock(set)` returns a block extracted from the set, the block is not removed from it.

Finally, we also implemented three functions for the map data type:

- `updateS(map, block, double, int, int, int)` returns the updated stake of the corresponding validator (index is passed as an integer parameter) with rewards and penalties applied based on whether they voted correctly for the block or not. The techniques for rewarding and penalizing validators are described in [21]. The function takes a double parameter representing the epoch for which the system is checking the votes, as well as the current state of the validator and the total stake of the system;
- `calcVotes(map, block)` returns the sum of the votes the block has received;
- `addVote(map, block, int)` returns an updated version of the map where the vote for the block has been recorded. The operation first checks if the map storing the votes for the block is empty. If it is not, it adds the stake of the validator who is voting to the corresponding index of the array. If the map is empty or the block the validator is voting for is not present in the map, it adds the block and the vote to the map.

```
1 private ExpressionMap addVote(ExpressionMap table, ExpressionBlock
    ↪ block, String validator){
2       int j = 0;
3       boolean found = false;
4       for(int i = 0; i<validator.length() and !found; i++) {
5          if(Character.isDigit(nameTmp.charAt(i))) {
6             j = i;
7             found = true;
8          }
```

```
9         }
10        String nameTmp2 = validator.substring(j);
11        int whichMiner = 0;
12        if(found) {whichMiner = Integer.parseInt(nameTmp2);}
13        boolean flag = false;
14        if(table.getVotedBlocks()!=null) {
15           for(int i = 0; i<table.getVotedBlocks().size(); i++) {
16              if(table.getVotedBlocks().get(i).getBlock().equals(block))
                    ↪ {
17                 flag = true;
18                 table.addVote(i,whichMiner,stake);
19              }
20           }
21           if(!flag) {
22              table.addBlock(block,whichMiner,stake);
23           }
24        }
25        else {
26           table = new ExpressionMap();
27           table.addBlock(block,whichMiner,stake);
28        }
29        return table;
30    }
```

Listing 4.12: The implementation of `addVote` operation.

4.2.3 Auxiliary Data Types

To model our data types in PRISM+, we created auxiliary data types. Specifically, we needed the `TreeList` data type to model the blockchain structure and a `Pair` to store the votes for the checkpoints in the Hybrid Casper protocol.

TreeList

The `TreeList` class is a proper ledger data structure since it represents the tree of blocks with all the informations needed. The fields of a `TreeList` are:

- `head`: the leaf block at maximal depth of the tree;
- `lastAdded`: the last block added to the blockchain;
- `lists`: the actual tree data structure (a list of list of blocks);
- `nodes`: the list containing all the blocks in the ledger in no particular order.

Like the `ExpressionBlockchain`, this data structure is always initialised with the genesis block as first element.

```
1  public class TreeList<T> {
2
3      private ExpressionBlock head;
4      private ExpressionBlock lastAdded ;
5      private ArrayList<ArrayList<ExpressionBlock>> lists = new ArrayList<
           ↪ ArrayList<ExpressionBlock>>();
6      private ArrayList<ExpressionBlock> nodes = new ArrayList<
           ↪ ExpressionBlock>();
7
8      public TreeList(ExpressionBlock head) {
9          this.head = head;
10         this.lastAdded = head;
11         nodes.add(head);
12         ArrayList<ExpressionBlock> tmp = new ArrayList<ExpressionBlock> ();
13         tmp.add(head);
14         lists.add(tmp);
15     }
```

Listing 4.13: The `TreeList` class.

Pair

The data type `Pair` is used to store the votes for each checkpoint in the Hybrid Casper protocol. It is a couple composed by a block and the array containing the votes for the block, which are represented by integers `(block, array of integers)`.

```
1  public class Pair {
2
3     private ExpressionBlock block;
4     private ArrayList<Integer> votes  = new ArrayList<Integer>();
5
6     public Pair(){
7        block = null;
8     }
```

Listing 4.14: The `Pair` class.

The votes are stored in such a way that the position of the array corresponds to the index of the voting node in the system.

```
1  public void setVote(int vote, int index) {
2        if(index>=votes.size()) {
3           for(int i=votes.size(); i<index; i++) {
4              votes.add(i,0);
5           }
6           votes.add(index,vote);
7        }
8        else {
9           votes.set(index,vote);
10       }
11    }
```

Listing 4.15: The `setVote` method.

The method `setVote` takes as parameters the index of the voting validator and the validator's stake (`vote`). If the index of the validator is greater than the size of the array, the array is filled with zeros because it means that not all validators with an index

less than the current one have already voted. Then, the vote of the current validator is stored in the correct index.

Chapter 5

The Bitcoin Protocol

This chapter discusses the models and results of analyses in PRISM+ for the Bitcoin protocol. In Section 5.1, the general model of the Bitcoin system is presented, with the assumption that all nodes function properly. Then, the PRISM+ model of a network with churning nodes (nodes that can leave and rejoin) is defined. Additionally, different network topologies (linear, ring, tree, and fully connected) are considered.

Then, we present the results of our analyses and compare the results we obtain when considering each setting. The first analysis, in Section 5.2, compares the model to the real Bitcoin network. The second one studies the trade-off between the security and the efficiency/scalability of the network, i.e. in Section 5.3 we study the interdependence between the probability of reaching a fork of length k and the difficulty of the cryptopuzzle. The last two analyses study the time needed to mine a new block and the probability of reaching a fork of length k for a network with churn miners and in networks using different kinds of topologies and are presented in Section 5.4 and 5.5, respectively.

In this chapter, for our analyses, we assume that all miners act honestly and do not engage in any malicious behaviour, such as double spending attacks or block withholding attacks [33].

5.1 Definition of the Models

In our model, a Bitcoin system is the result of the parallel composition of n `Miner` processes, n `Hasher` processes and a process called `Network`. `Hasher` processes model the attempts of the miners to solve the cryptopuzzle, while the `Network` process model the broadcast communication among miners.

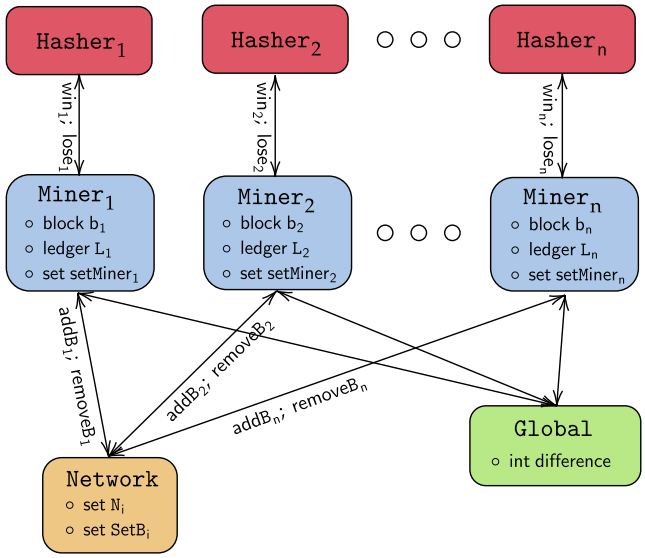


Figure 5.1: The Bitcoin model architecture.

At the beginning, we assume that miners are connected through a network guaranteeing broadcast. Later on, we shall consider other topologies (see Section 5.1); the actual architecture is illustrated in Figure 5.1. The abstraction also uses an auxiliary process, called `Global`, that computes the length of forks, see Section 5.2. As already said, in order to abstract out the solution of the cryptopuzzle and the broadcast of new blocks, we use rates.

General Model

For the sake of clarity, we present a simplified version of the `PRISM+` code implementing our processes. The actual abstraction, the analysed properties with tests and the instructions for the use of the library are available on the online repository[1].

[1] `https://github.com/adeleveschetti/bitcoin-analysis`

```
1 module Hasher_i
2    Hasher_STATE_i : 0;
3
4    [win_i] (Hasher_STATE_i=0) ->mR : Hasher_STATE_i'=0 ;
5    [lose_i] (Hasher_STATE_i=0) ->lR : Hasher_STATE_i'=0 ;
6 endmodule
```

Listing 5.1: Simplified model of an Hasher.

A Hasher process is defined in Listing 5.1. It represents the PoW algorithm performed by miners: those miners who want to solve the cryptopuzzle synchronise with the Hasher which "answers" telling them if they succeeded or not.

The Hasher consists of two transitions: the first one with action `[win_i]` and rate `mR` is triggered when the synchronising miner finds a solution for the PoW (`mR` is taken such that $0 < \texttt{mR} < 1$); the second one with action `[lose_i]` and rate `lR` ($\texttt{lR} = 1 - \texttt{mR}$) is triggered when the synchronising miner does not find a solution to PoW. In both cases the Hasher process makes a silent action.

A Miner, described in Listing 5.2, has a ledger, called $\texttt{L}_i$, a set containing the blocks to be added to the ledger, called $\texttt{setMiner}_i$, a block $\texttt{b}_i$ and a integer $\texttt{c}_i$. The variable $\texttt{b}_i$ is used to store the block the Miner creates and to store the newly extracted block from the set. The integer $\texttt{c}_i$ is a counter whose value ranges between 0 and 100 (initially is zero) and which keeps track of the number of blocks created by the Miner, so that we can assign unique names to blocks.

The Network process is defined in Listing 5.3. It contains a set of blocks $\texttt{set}_i$ for each $\texttt{Miner}_i$ that represents the messages to be delivered to the miner. The set $\texttt{N}_i$, in the case of the broadcast topology, is equal to $\{0, \cdots, i-1, i+1, \cdots, n-1\}$, e.g. the indexes of the miners to whom a block must be sent (line 7). Also, the Network process contains a transition for each Miner to model sending and receiving messages. More precisely, the Network synchronises with the Miner who won the PoW (in the winner state) using the `addB_i` action. As a result of this synchronisation, Network updates the sets of blocks of the miners contained in the set $\texttt{N}_i$. [2]

[2] Actually, the set $\texttt{N}_i$ is not present in the PRISM+ model. The actions are replicated for all the Miners without the for loop.

```
module Miner_i
    STATE_i : [Mine,Winner,Lost,Add,Move] init Mine;
    b_i : block {Miner_i,0;genesis,0};
    L_i : ledger ⟨{genesis,0;genesis,0}⟩;
    c_i : [0..100] init 0;
    setMiner_i : set [];

    [win_i] (STATE_i=Mine) ->
         hR_i : (b_i'=createB(Miner_i,c_i,L_i))&(c_i'=c_i+1)&(STATE_i'=Winner);
    [lose_i] (STATE_i=Mine) -> hR_i : (STATE_i'=Lost);
    [addB_i] (STATE_i=Winner) -> 1 :(L_i'=addBtoL(L_i,b_i))&(STATE_i'=Mine);
    [] (STATE_i=Lost) & !isEmpty(set_i) -> 1 : (b_i'=receive(set_i))&(STATE_i'=Move);
    [] (STATE_i=Lost) & isEmpty(set_i) -> 1 : (STATE_i'=Mine);
    [] (STATE_i=Lost) & !isEmpty(setMiner_i) ->
                    1 : (b_i'=receive(setMiner_i))&(STATE_i'=Add);
    [] (STATE_i=Lost) & isEmpty(setMiner_i) -> 1 : (STATE_i'=Mine);
    [removeB_i] (STATE_i=Move) ->
                    1 : (setMiner_i' = add(setMiner_i, b_i))&(STATE_i'=Mine);
    [] (STATE_i=Add)&(canBeIns(L_i,b_i)) ->
      1 : (L_i'=addBtoL(L_i,b_i)&(setMiner_i'=remove(setMiner_i,b_i))&(STATE_i'=Mine);
    [] (STATE_i=Add)&(!canBeIns(L_i,b_i)) -> 1 : (STATE_i'=Mine);
endmodule
```

Listing 5.2: Simplified model of a Miner.

```
module Network
    n : numberOfMiners
    for i from 0 to n-1:
        set_i : set [];
        N_i : set [];
    for i from 0 to n-1:
        [addB_i] -> r_b : foreach k in N_i { set_k'=add(set_k,b_i); };
    for i from 0 to n-1: [removeB_i] -> 1 : set_i' = remove(set_i,b_i);
endmodule
```

Listing 5.3: Simplified model of the Network.

Below we describe in detail how our processes abstract the Bitcoin protocol. As the reader can observe from the code in Listing 5.2, `Miner`'s state is initially set to `Mine`. In this state it can synchronise with $\texttt{Hasher}_i$ using either the `win_i` or `lose_i` actions. As already said, this synchronisation abstracts the cryptopuzzle solution. Note that the time needed for the creation of a new block at miner i is a random number sampled from an exponential distribution with rate proportional to the ratio of the difficulty of the PoW problem and the hashing power of the miner. Therefore, the chosen action, `win_i` or `lose_i`, depends on the difficulty of the problem (represented by the hasher values `mR` and `lR`) and on the hashing power of the miner (represented by $\texttt{hR}_i$). Since the rate of a synchronisation is equal to the product of the rates of the two actions, the rate of mining a new block is $\texttt{mR}\times\texttt{hR}_i$, which corresponds to the parameter λ_{m_i} introduced in Section 2.2.1. Whereas the rate of losing the competition is $\texttt{lR}\times\texttt{hR}_i$.

If the miner wins, it changes its status in `Winner` (lines 8 and 9), updates its ledger and sends the new block to the Network (action `addB_i` at line 11) in order to forward it to the other miners (i.e. updating other miners' sets with the new block). This action is taken with a certain rate $\texttt{r}_b$, which simulates the latency of the network and corresponds to the product between the rate 1 (for the action of the Miner) and the rate $\texttt{r}_b$ of the Network action (line 7 of Listing 5.3). If the system is in a state of fork, the Miner will always create the new block following the longest chain rule: by choosing the chain with the most work (in our implementation, it is always the longest chain).

If the miner loses, its status becomes `Lost` (line 10) and it checks for new blocks in the Network process.

If there are new blocks, the miner chooses randomly one of them (with the operation `extractBlock()`). This random choice simulates the delay due to the topology of the network. In our model, the rate $\texttt{r}_b$ and the random selection of blocks from the sets simulate the communication delay of messages in the Bitcoin network.

Then, the state of the Miner becomes `Move` and the Miner adds to its local set $\texttt{setMiner}_i$ the block $\texttt{b}_i$ (lines 17 and 18). Moreover, the Miner synchronises with the process Network with action `removeB_i`. The Network removes the block $\texttt{b}_i$ from the set of the Miner $\texttt{set}_i$. Then the state of the Miner becomes `Mine`.

Otherwise, the Miner can try to pick a block from its local set (lines 14-15) and, in

this case, a block is randomly extracted from the local set $\texttt{setMiner}_i$. If the block taken from the local set can be added in the ledger (which means that the function $\texttt{canBeIns(L}_i\texttt{,b}_i\texttt{)}$ returns `True`), the Miner adds the block to its ledger and removes it from the local set (lines 19-20). Finally, its status is set to `Mine` and the process starts again. Otherwise, the block is not removed from the local set and the process starts again (line 21). If both the local set and the set stored in the Network process are empty, the Miner does nothing and its status returns `Mine` (lines 13 and 16). The time spent in performing these actions is simulated by the rate 1. This rate is much higher than the other rates (which are numbers in the $[0, 1]$ interval) because it corresponds to local management operations of the Miner. Therefore, the probability that a Miner tries to add a received block in its ledger is higher than the probability of receiving or mining a new block.

It is worth noticing that a block is added in the correct position of the ledger, even if it is a *stale block*. In our model stale blocks are represented as valid blocks which are not part of the blockchain. In contrast, an orphan block is modelled as a block received by a miner, but that does not have its entire ancestry (yet) in the local ledger and thus cannot be added. So an *orphan block* is not added to the ledger and is left in the local set $\texttt{setMiner}_i$.

Churn Nodes

Nodes that may leave the Bitcoin network and rejoin after some time are called *churn nodes*. As described in [61, 60], while a node is away from the network, other active nodes continue processing transactions, mining and adding blocks to their respective ledgers. When a node rejoins the network, its ledger is out of date and needs to be updated before the node can take part in network activities. Therefore, the first action to be taken after rejoining is to download all blocks that were added to the set of the Network during its sleep. When the blocks have been downloaded, the miner can start adding them to the ledger. In the model of Listing 5.2, this is performed by transiting to the state `Mine`. In order to model churn miners, we define a controller process that awakes and shuts down miners following a given policy explained below. The uncertainty is modelled by rates and the controller consists of a sequence of states that alternate `awake` and

sleep synchronisations with the corresponding miner. Listing 5.4 shows a controller $\texttt{Controller}_i$ for a miner $\texttt{Miner}_i$ with 11 states. Note that the controller is a finite state system that, when the number of states are even, will leave the corresponding miner active forever; when the number is odd, it will leave the miner inactive forever. It is easy to define alternative controller with cyclic behaviours.

```
1  module Controller_i
2    Controller_STATE_i : [s0,s1,. . . , s10] init s0;
3
4    [sleep_i] (Controller_STATE_i = s0) -> r_i0 : Controller_STATE_i' = s1 ;
5    [awake_i] (Controller_STATE_i = s1) -> r_i1 : Controller_STATE_i' = s2 ;
     ...
12   [sleep_i] (Controller_STATE_i = s8) -> r_i8 : Controller_STATE_i' = s9 ;
13   [awake_i] (Controller_STATE_i = s9) -> r_i9 : Controller_STATE_i' = s10 ;
14 endmodule
```

Listing 5.4: Model of a controller with 11 states.

```
1  module Miner_i
2     STATE_i : [Mine,Winner,Lost,Add,Move,Update,MoveUpdate,Sleep] init Mine;
3     b_i : block {Miner_i,0;genesis,0};
4     L_i : ledger ⟨{genesis,0;genesis,0}⟩;
5     c_i : [0..100] init 0;
6     setMiner_i : set [];
7
8     [sleep_i] (STATE_i = Mine) -> 1 : (STATE_i' = Sleep);
      ...
32    [addB_i] (STATE_i=Update) & !isEmpty(set_i) ->
33                  1 : (b_i'=receive(set_i))&(STATE_i'=MoveUpdate);
34    [removeB_i] (STATE_i=MoveUpdate) ->
35                  1 : (setMiner_i'=add(setMiner_i, b_i))&(STATE_i'=Update);
36    [] (STATE_i=Update) & isEmpty(set_i) -> 1 : STATE_i'= Mine ;
37    [awake_i] (STATE_i = Sleep) -> 1 : (STATE_i' = Update);
38 endmodule
```

Listing 5.5: Simplified model of a *dynamic* miner.

The churn miner of Listing 5.5 extends the one of Listing 5.2 with three additional states: `Sleep`, `MoveUpdate`, and `Update`.

As before, the initial state of this Miner is `Mine` where, in addition, it may synchronise with the controller (action `sleep_i`) and, after a certain amount of time, modelled by the exponential parameter r_{i0}, the Miner state becomes `Sleep` (line 8). In this state, the miner may synchronise with the controller again (action `awake_i`) and its state becomes `Update` (line 32).

In the `Update` state, the Miner synchronises with the `Network` process and extracts all the blocks from the corresponding set in `Network` by moving them into its local set $\texttt{setMiner}_i$ (lines 34-35). When $\texttt{set}_i$ becomes empty (line 36) the Miner state is set to `Mine` and the Miner can resume its standard behaviour, which is the one defined in Section 5.1 in Listing 5.2 (lines 8-21).

In our simulations we consider three controllers: one with four states (so sleep-awake-sleep synchronisations, the first sleep has a very high rate, therefore the corresponding miner goes asleep immediately), the second with two states (one sleep synchronisation only: when the miner shuts down, it will be down forever) and the third one with five states (sleep-awake-sleep-awake synchronisations). Additional experiments with larger number of churn miners and with cyclic behaviour are left to future work.

Network Topologies

We modelled several kinds of network topologies and analysed how they affect the likelihood of a fork. Network topology refers to how nodes are connected with each other and transmit new blocks.

We study three different network topologies: the ring topology, the tree topology and the linear topology. The three topologies have been modelled by changing only the Network process; Hasher and Miner are those defined in Section 5.1. Listing 5.6 shows the modified code for the Network process. As the reader can observe, the code is the same as of the one presented in the Section 5.1, except for line 10. In particular, we modified the set $\texttt{N}_i$ containing, for each node, the set of nodes to whom the new blocks have to be forwarded. When a Miner extracts a block received by the Network, the block is forwarded to the nodes contained in the set $\texttt{N}_i$. This set is defined according to the

topology as follows:

- *Linear topology* (Figure 5.2a): $\mathtt{N}_i$ contains the previous node and the next one (except for the terminal nodes where $\mathtt{N}_i$ are singletons). For instance, for miner $\mathtt{Miner}_i$:

$$\mathtt{N}_i = \{\mathtt{Miner}_{i-1}, \mathtt{Miner}_{i+1}\} \text{ if } \mathtt{i} \neq \mathtt{0,n}$$

$$\mathtt{N}_0 = \{\mathtt{Miner}_1\}$$

$$\mathtt{N}_n = \{\mathtt{Miner}_{n-1}\}$$

- *Ring* (Figure 5.2b): the set $\mathtt{N}_i$ contains the previous and the next node for each miner. Thus, for every miner

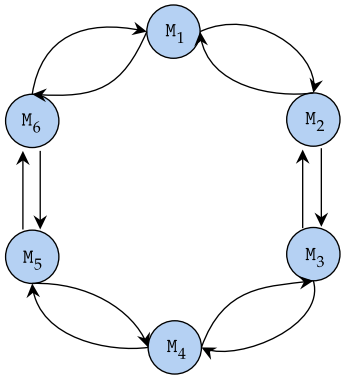
$$\mathtt{N}_i = \{\mathtt{Miner}_{((i-1)\%n)}, \mathtt{Miner}_{((i+1)\%n)}\}.$$

- *Tree topology* (Figure 5.2c): every $\mathtt{N}_i$ contains the parent node and the children node, except for the root of the tree and the leaves. Roots have only children nodes; leaves have only the parent.

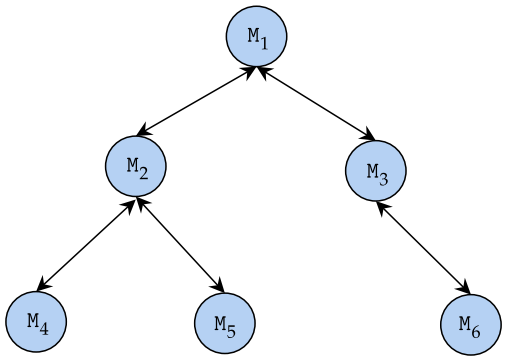
```
1  module Network
2      n : numberOfMiners
3      for i from 0 to n-1:
4          set_i : set [];
5          N_i : set [];
6      for i from 0 to n-1:
7          [addB_i] ->  r_b : foreach k in N_i { set_k'=add(set_k,b_i); };
8      for i from 0 to n-1:
9          [removeB_i] -> 1 : set_i' = remove(set_i,b_i)
10                          & foreach k in N_i { set_k'=add(set_k,b_i); };
11 endmodule
```

Listing 5.6: Simplified model of the Network.

(a) Linear topology.



(b) Ring topology.



(c) Tree topology.

Figure 5.2: Network topologies.

The three topologies have been selected because they are the simplest to realise in practice. The goal is to compare the resilience to forks of these topologies with respect to the broadcast topology (where every miner forwards the block to all the other miners). Henceforth, one can choose the best topology according to the preferred trade-off between risks of forks and connection costs.

5.2 Coherence of the Model

As previously explained in Section 3.3, in order to analyse the system and check the properties we want to study, we define in PRISM+ the properties of interest in the stochastic temporal logic CTL. For example, the formula

```
P=?[F<=T "winner"]
```

defines the probability that some miner mines a new block within the first `T` time units. Thus, when checking the above property, PRISM+ must check whether `F<=T "winner"`

is true for each path. This is the formula we use to assess the coherence of our model with respect to Bitcoin – see Section 5.2 – since it allows us to study the probability of creating a new block by varying the time.

Another example is the formula that checks the occurrence of forks between the blockchains in the network. To formalise this formula we introduced in our model a suitable module to compute forks, called `Global`, whose code is reported in Listing 5.7.

```
1 module Global
2     difference : [0..100] init 0;
3
4     [] (STATE_1 = Add) | ... | (STATE_n = Add) ->
5             1 : (difference' = calculateFork(L_1, ..., L_n));
6 endmodule
```

Listing 5.7: The Global process.

The process `Global` computes the difference between the blockchains of the system every time a ledger is modified, e.g. when the $\texttt{Miner}_i$ changes its state to `Add`, using the `PRISM+` operation `calculateFork` (see Section 4.2.2). The value returned by `calculateFork` is stored in the variable `difference`. Therefore, the probability of reaching a state of fork of length k within the first `T` time units is defined as:

```
P=?[F<=T difference = k]
```

The complete definition of the considered properties can be accessed in the on-line repository [13]. For sake of completeness, we point out that all the analyses presented in the following sections and in Chapter 7 have been carried out on a Virtual Machine with 8 VCPU and 64 GB RAM.

To validate our model with respect to Bitcoin, we take some well-known values of the protocol from the literature and we compare them with the results of our simulations. We begin by studying mining rates. According to the hash rate distribution of Bitcoin mining pools on May 2020, the probability that a block is mined in Bitcoin within 600 seconds is about 63% (or $1 - e^{-1}$). In 30 minutes (1800 seconds) a block has about 95%

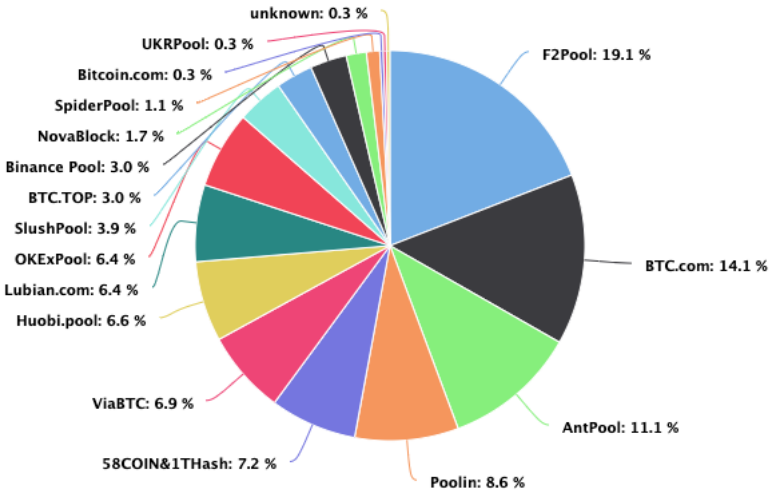


Figure 5.3: Hashrate distribution of Bitcoin mining pools on May 2020. Source: `https://www.blockchain.com`

chance of being found and in 3000 seconds the probability that someone has found the block is close to 1 [3]. In our system, we have a model with 13 miners. Thus, we decided that each miner corresponds to the hashing power distribution of (the main) Bitcoin pools as illustrated in Figure 5.3. With these values, we obtain a probability of mining a new block which varies over time as shown in Figure 5.4.

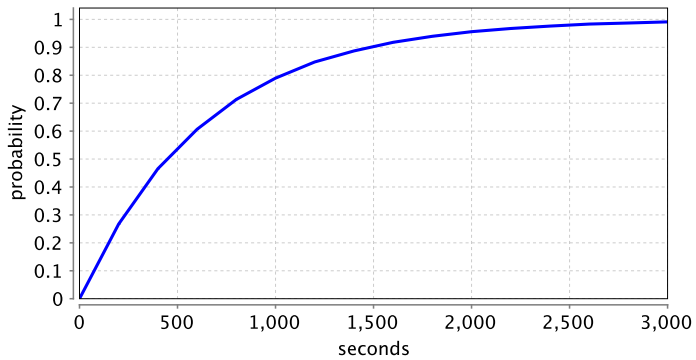


Figure 5.4: Probability of mining a block.

In particular, Figure 5.4 displays the probability that some miner in the system has mined a new block and, as discussed before, this analysis is a sanity check to verify that

[3] `https://en.bitcoin.it/wiki/Confirmation`

our model is coherent with Bitcoin.

The plot shows two important facts:

(i) the probability of mining a new block has an exponential behaviour as expected from the literature;

(ii) the timings are in line with those of the protocol: indeed, the probability that a block is mined in 3000 seconds is almost 1. Similarly, the probabilities at 600 and 1800 seconds are coherent with the protocol.

If the first fact is not surprising for how we built the model, the second one makes us confident that the results obtained by our experiments below are meaningful, because these probabilities are coherent with what presented in the literature.

Then, we study the probability of reaching a state with a fork of length 1 by varying the communication delay. The expected output is that the higher is the rate for the communications (which means the faster is the communication between the nodes of the network), the lower should be the probability of the fork.

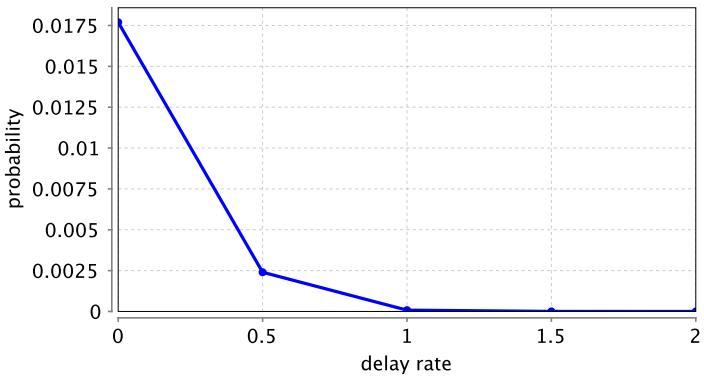


Figure 5.5: Probability of reaching a fork of length 1 by varying the broadcast delay; the bound time `T` is set to 600 seconds.

In fact, this is what Figure 5.5 highlights. In particular, with the analysis reported in Figure 5.5, the goal was to check if the probability of reaching a state of fork obtained in our model was coherent with the one presented in the literature and, moreover, if the exponential parameter was coherent with the one that can actually model the network delay in Bitcoin. From Figure 5.5 what can be seen is that, when the communication

rate is $r_b = 1/12.6 = 0.08$, we obtain results in line with Bitcoin, as presented in [27], since the probability is 0.0175. Moreover, the parameter that allows us to obtain this result, is in line with what happens in the reality. In fact, from the literature, we know that the time needed for a block to be received by all the miners in the network can be approximated as an exponential distribution with parameter 1/12.6.

Thus, as a last analysis for validating our model, we study the probability of having forks of increasing length (forks of length $k > 1$) when the broadcast rate is fixed to $r_b = 0.08$. The results of our simulations are in Figure 5.6. The reader can observe

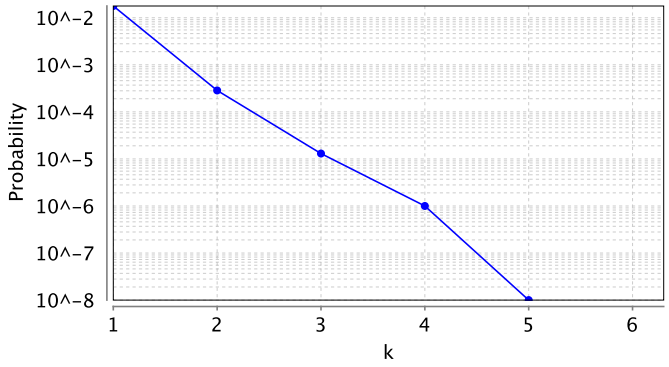


Figure 5.6: Probability of a fork of length k; the bound time $\mathtt{T}$ is set to $k * 600$ seconds.

that the probability to have a fork of length 5 is of the order of 10^{-8}, whereas it is approximately zero when the length of the fork reaches 6. This is a key result because blocks at depth 6 are considered as confirmed in Bitcoin and therefore paid and, having a fork of length 6 so unlikely, means that the confirmation time guarantees the security of the transactions.

5.3 Variation of Cryptopuzzle Difficulty

In this section, we start our study of the resilience of the Bitcoin protocol to relevant changes of the rates by changing the rates of creating blocks.

We begin by analysing the probability of having a fork while varying the difficulty of the cryptopuzzle (in Bitcoin this difficulty is adjusted with respect to the computational power of the miners, in order to have a new block on average every 10 minutes). Figure 5.7

highlights the relationship between the probabilities of mining a block within a specific amount of time with two two different average mining rates, denoted with $1/\tau$ in the figure. The comparison is between a system with Bitcoin average mining rate (1/600)

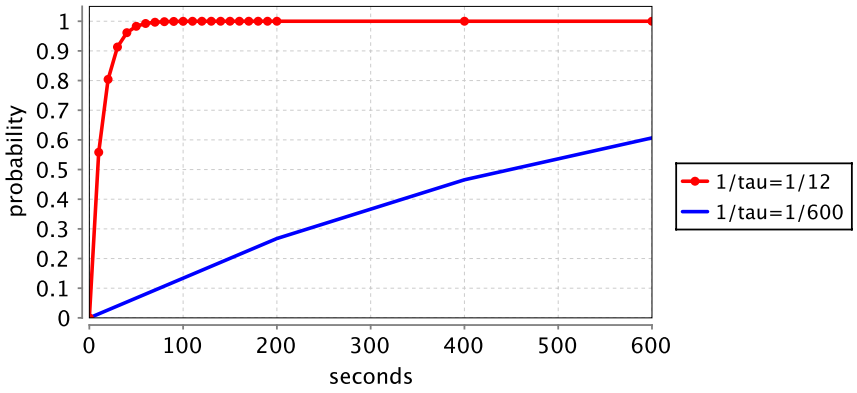


Figure 5.7: Probability of mining a block within 600 seconds.

and a system where a new block is produced every 12 seconds (1/12). Of course, the probability that a miner finds a new block in the second system is much higher than Bitcoin. In particular, after 100 seconds the probability that a miner mines a new block is 1 when the average mining rate is 1/12; on the contrary, with the Bitcoin rate, the probability is less than 0.2. Figure 5.8 shows the relationship between the length of the

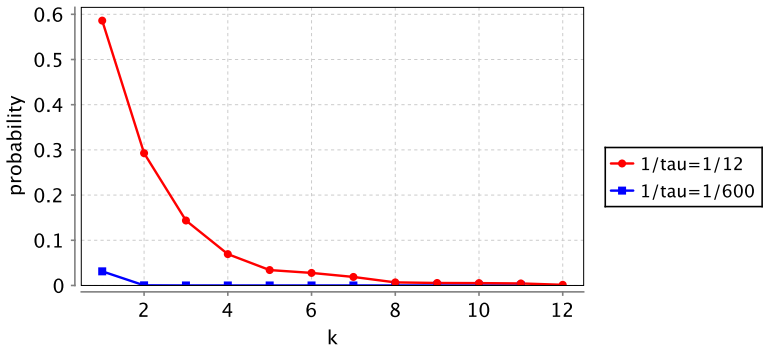


Figure 5.8: Probability of a fork of length k; the bound time T is set to 600 seconds.

forks and the average mining rates. In this case, the probability of reaching a fork of length 6 with average mining rate 1/12 is greater than 0, whereas it becomes zero with the average mining rate of Bitcoin. This is due to the fact that if it is way easier to create

a new block, then it is more likely that two or more miners create a block at the same time. Of course, a system with a mean time of creating a block of 12 seconds should choose a higher number of confirmations, otherwise the system would not guarantee the safety of the transactions.

Finally, we study how the time required to mine a block varies when we consider different cryptopuzzle difficulties. Since the difficulty of the cryptopuzzle is inversely proportional to the rate of the mining process, one might be interested in studying the trade-off between speed and security.

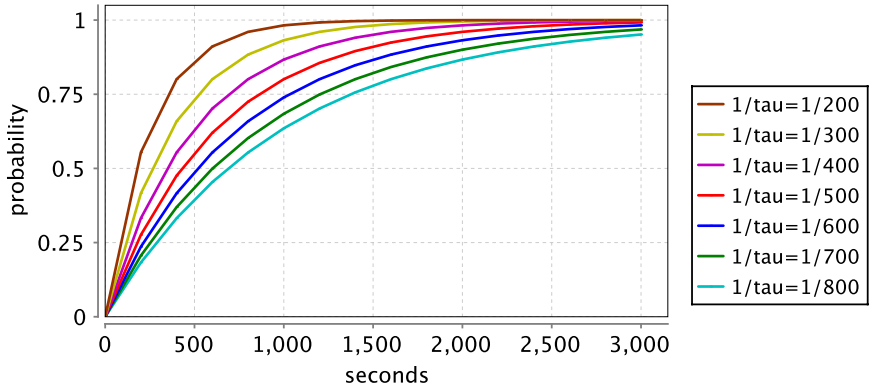


Figure 5.9: Probability of mining a block within 3000 seconds.

The results of the simulations in Figure 5.9 are straightforward and confirm that the easier the cryptopuzzle is, the faster the entire system mines a new block. This follows from the fact that the average time τ required to mine a new block is given by

$$\tau = \frac{2^{32} D}{H}$$

where D is the cryptopuzzle difficulty and H is the global hash rate. Figure 5.10 displays how the probability of reaching a fork of length 1 varies depending on different average mining intervals. Obviously, the probability decreases while the parameter D increases. Our results show that a good balance between speed and safety can be obtained with a mining rate equal to 1/500 per second. Indeed, with this rate, the process of mining a block is faster than in Bitcoin (1/600), but the probability of reaching a state of fork is not much higher. Even if this is a theoretical result, it shows that a better trade-off

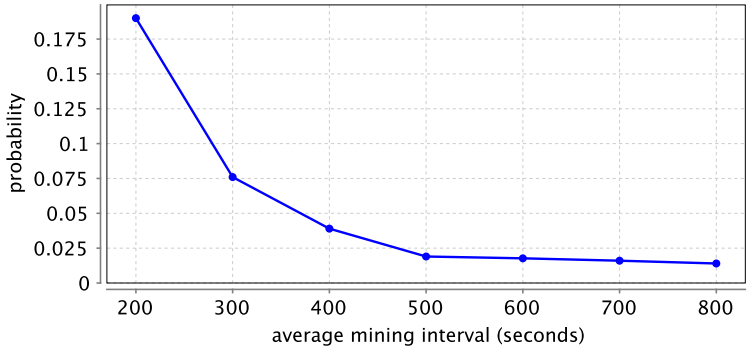


Figure 5.10: Probability of a fork of length 1 versus the average mining interval; the bound time `T` is set to 600 seconds.

between the rate of the mining process and the security of the network can be obtained and can be measured. Of course, changing this trade-off may impact the behaviour of Bitcoin in different aspects. Since mining is a energy consuming task, one may expect that speeding up the process may lead to some energy savings in practice. Although this seems reasonable in theory, actually, it also depends on the behaviour and strategies of miners, e.g., they may decide to invest more in mining since it is now "easier" to mine new Bitcoins. It is not easy to predict with certainty how this change impacts miners' strategies. We leave studying this problem as a future work. A related aspect concerns understanding how the value of Bitcoins in the market varies, when the cryptopuzzle difficulty changes. Actually, a recent study [34] seems to suggest that the hash rate is not useful in predicting the Bitcoin price on its own. However, we believe that the change could affect the fees that miners receive for their work, so impacting their strategies. Also, studying this problem is left as a future work.

5.4 Churn Nodes

In this section, we focus on the simulations of a system using the broadcast topology (see the `Network` process in Listing 5.3) but with three churn nodes. As anticipated in Section 5.1, our model consists of 13 "static" miners and three churn miners. The first miner goes asleep as soon as the process starts and then awakes after a while. The other two miners, at the beginning, participate at the mining process, but shut down after a

given amount of time. The difference between the two is the fact that one of them, when it shuts down, does it forever, whereas the other awakes again after a while. A churn miner impacts on the Bitcoin protocol because, when a node leaves or joins the network, the overall hashing power changes [59, 60, 61].

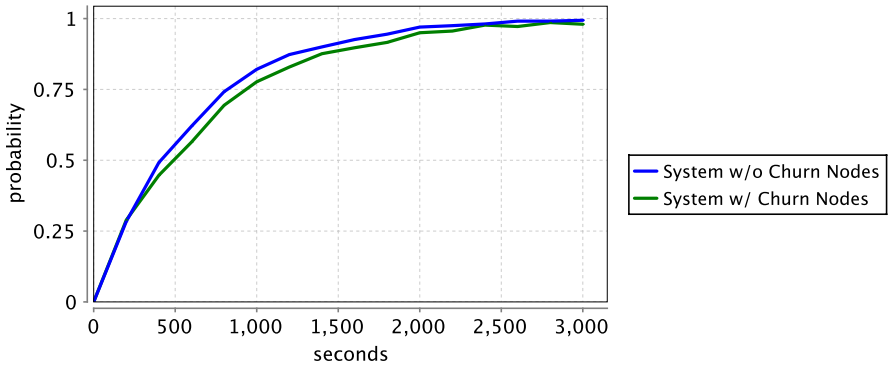


Figure 5.11: Probability of mining a block within 3000 seconds.

This remark is confirmed by Figure 5.11, which compares the time needed to mine a new block with the presence of churn nodes to the time needed in the Bitcoin system with only static nodes. Since in the dynamic system, there are less nodes trying to solve the cryptopuzzle, the probability that some miner wins is lower. Also the probability of reaching a state of fork is lower as reported in Figure 5.12. This is due to the fact that there are less miners and, in this setting, each miner is connected to every other, thus the presence of churning nodes does not lead to message losses.

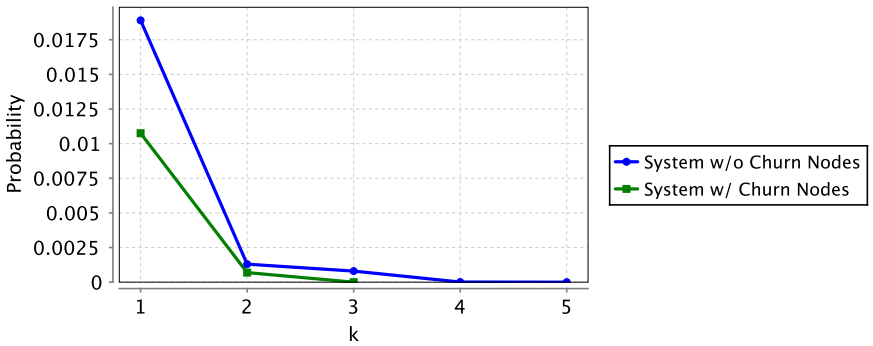


Figure 5.12: Probability of a fork of length k, the bound time T is set to $k * 600$ seconds.

Finally, we study the latency of a churn miner when it rejoins the network because it has to download all the blocks that were mined during its absence. In particular, we assume that the mean node sleep rate ranges from 2 to 10 hours, as suggested in [61, 60]. Our experiments highlight that the synchronisation process requires little time, as the reader can observe from Figure 5.13 which shows the probability that a node (with different sleep rates) synchronises the missing blocks within a minute after rejoining the network.

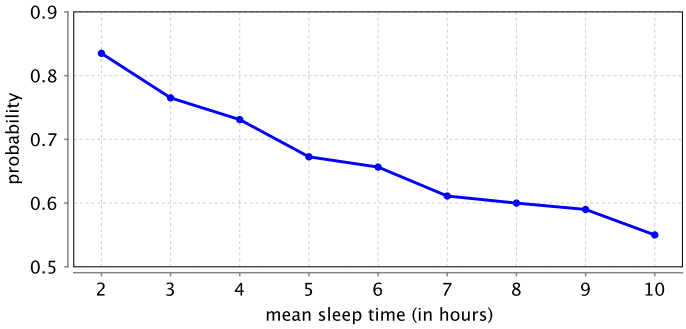


Figure 5.13: Probability that the node can synchronise in a minute with mean sleep time ranging from 2 to 10 hours.

Clearly, the longer the nodes are down the lower the probability that it quickly synchronises because the number of blocks mined during its sleeping period increases.

5.5 Different Topologies

In Section 5.1, we described the models of different topologies. The results of our experiments are presented thereafter. Please note that the rates r_b presented in Section 5.1 are set to 1 in the simulations of linear, tree and ring topologies. This is because we no longer need to approximate the network latency, and we want to understand the behavior of the system when the nodes are not fully connected. In this setting, we do not model the latency of the network using rates, but rather use the set of connected nodes. Specifically, we set r_b to 1 to consider it as an instantaneous action (compared to other actions with lower rates).

In Figure 5.14 we show that the probability of mining a block is not affected by

changing the network topology. This outcome is trivial because we are changing how the

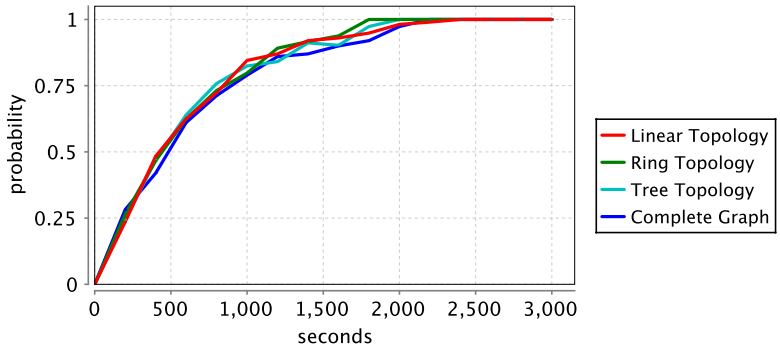


Figure 5.14: Probability of mining a block within 3000 seconds.

nodes receive and send the new blocks and it does not affect the time needed to create a new block, which remains unchanged.

It turns out that the main difference between topologies is the probability of reaching a fork of length k. When a new block is mined, if it is not distributed to all nodes but

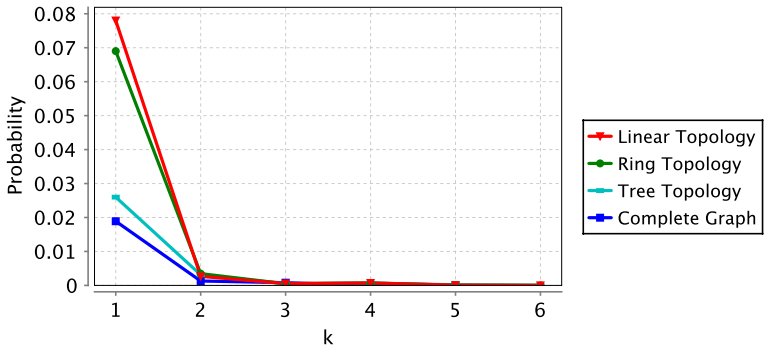


Figure 5.15: Probability of a fork of length k, the bound time `T` is set to $k * 600$ seconds.

only a subset, the likelihood of forks increases. Figure 5.15 compares the four topologies: broadcast, linear, ring, and tree topologies. Our results show that the smaller the subset of nodes that receive the block per miner, the higher the probability of a fork.

Chapter 6

A Formal Analysis of the Bitcoin Protocol

The purpose of this chapter is to show the results obtained by analysing the Bitcoin protocol through a stochastic transition system. The PRISM+ system allows us to compute the probability of devolving into a "larger inconsistency", e.g. transiting from a state with a fork of length m to a state with a fork of length $m + 1$. This work, which has required a time-consuming analysis of the stochastic transition system, has given a formula that is parametric with respect to the number of nodes, their hashing power and the latency of the network.

In Theorem 6.1, we compute the probability that ledgers turn into a state with more severe inconsistencies, e.g. with longer forks, under the assumption that messages are not lost and nodes are not hostile. Moreover, in Theorem 6.2, we study how the presence of hostile nodes trying a double spending attack mining blocks in wrong positions impacts the consistency of the ledgers.

6.1 Honest Miners

The main goal of this chapter is to compute the probability of the Bitcoin system defined in Listing 5.2 to devolve into states of fork by analysing the transition system. In order

to ease our arguments, among the possible states of the stochastic transition system obtained from the model, we select those where the blocks have all been delivered, we call it a *completed* state.

Definition 6.1. *A state of a Bitcoin system is called* completed *when there is no block to deliver (every* $\texttt{set}_i$ *in the* `Network` *process is empty) and the blocks in the local sets of* $\texttt{Miner}_i$ *have already been inserted in the corresponding ledgers (every* $\texttt{setMiner}_i$ *in* $\texttt{Miner}_i$ *is empty).*

This scenario is usual in Bitcoin because the rate of block delivery is much higher than the one of mining. For example, the nodes that have not yet received the last block after 40 seconds are less than 5%, whilst blocks are mined every 10 minutes [27]. If the system is in a completed state, it follows that if the paths from the last valid leaf to the genesis block of two ledgers differ, it means that the system is in a state of fork. We can formalise that with the following proposition, recalling the definition of the handle (Definition 2.1).

Proposition 6.1. Let $\texttt{L}_1{=}\langle \texttt{T}_1; \texttt{p}_1 \rangle$ and $\texttt{L}_2{=}\langle \texttt{T}_2; \texttt{p}_2 \rangle$ be two ledgers of different nodes. Therefore, if $\texttt{L}_1 \neq \texttt{L}_2$ and $\texttt{T}_1 = \texttt{T}_2$ then $\texttt{handle}(\texttt{L}_1) \neq \texttt{handle}(\texttt{L}_2)$.

An inconsistent state of the ledgers can be expressed as the maximum number of blocks that differ between two (or more) blockchains, as presented in Definition 2.3. We now have everything we need to go into the details of our theorem. For the sake of simplicity, in the rest of the chapter:

- we shorten $\texttt{mR}{\times}\texttt{hR}_i$ into r_{w_i};
- the rate r_b represents the time needed to deliver a block within the network;
- the rate r that corresponds to local management operations by miners is approximated to 1 because the other rates are in the interval $[0, 1]$.

Theorem 6.1. Let S be a completed state of a Bitcoin system consisting of n miners with ledgers $\mathtt{L}_1{=}\langle \mathtt{T}_1; \mathtt{p}_1\rangle$, ..., $\mathtt{L}_n{=}\langle \mathtt{T}_n; \mathtt{p}_n\rangle$, respectively, such that $\mathtt{L}_1 = \ldots = \mathtt{L}_k$ and $\mathtt{L}_{k+1} = \ldots = \mathtt{L}_n$ and $\mathtt{L}_1 \neq \mathtt{L}_{k+1}$. Let $\mathtt{L}_1$ and $\mathtt{L}_{k+1}$ have fork of length m. Then the probability $P_{(m+1)_{fork}}$ to reach a completed state with fork of length $m+1$ is smaller than

$$\sum_{\substack{1\leq i\leq n \\ H\subset\{1,\ldots,n\}\setminus\{i\} \\ i\leq k \Rightarrow j\in\{k+1,\ldots,n\}\setminus H \\ i>k \Rightarrow j\in\{1,\ldots,k\}\setminus H}} \Theta(i,|H|,j) \tag{6.1}$$

where $R = \sum_{j=1}^{n} r_{w_j}$ and

$$\Theta(i,\ell,j) = \frac{r_{w_i}}{R}\frac{r_{w_j}}{(R+(n-1-\ell)r_b)}\prod_{1\leq h\leq \ell}\frac{h\ r_b}{R+(n-h)r_b}\prod_{1\leq a\leq 2n-2-\ell}\frac{a\ r_b}{R+a\ r_b}\,.$$

Before going into the details of the actual proof, we give just a hint of how it works. To explain the probability, assume to have a fork of length 1 due to miners having equal ledgers (since the state is completed) and two different handlers. Let $1,\ldots,k$ be the nodes with one blockchain and $k+1,\ldots,n$ be the nodes with the other blockchain. At this point, we must consider four distinct categories of events and, correspondingly, four probabilities.

I. Assume that a node $1 \leq i \leq k$ mines a new block; the probability will be $\dfrac{r_{w_i}}{R}$.

II. The new block is then communicated to a set H of nodes that immediately add it to the local ledger. This operation happens with probability

$$\left(\prod_{1\leq h\leq |H|}\frac{h\ r_b}{R+(n-h)r_b}\right).$$

III. At this point, in order to obtain a fork of length 2, a node $j \in \{k+1,\ldots,n\}\setminus H$

must mine a block as well. The probability of this operation is

$$\frac{r_{w_j}}{R + (n - 1 - \ell) r_b}.$$

IV. Finally, every node receives the two mined blocks, this process has probability

$$\left(\prod_{1 \leq a \leq 2n-2-\ell} \frac{a\ r_b}{R + a r_b} \right).$$

It is straightforward that the same result can be obtained if the first node that mines a block belongs to the other partition ($j \in \{k+1, \ldots, n\}$). Henceforth, the probability to reach a completed state with fork of length 2 from the initial state is

$$\sum_{\substack{1 \leq i \leq k \\ H \subset \{1,\cdots,n\} \setminus \{i\} \\ j \in \{k+1,\cdots,n\} \setminus H}} \Theta(i, |H|, j) + \sum_{\substack{k+1 \leq j \leq n \\ H \subset \{1,\cdots,n\} \setminus \{j\} \\ i \in \{1,\cdots,k\} \setminus H}} \Theta(j, |H|, i)$$

which is exactly what stated in the theorem. In the first part of the proof, it is shown that the formula is less than 1. In the second part, it is established that the formula is an upper bound for $P_{(m+1)_{fork}}$.

Proof. **Part 1: the formula** (6.1) **is smaller than 1.** The formula (6.1) can be rewritten

$$\sum_{\substack{1 \leq i \leq k \\ H \subset \{1,\cdots,n\} \setminus \{i\} \\ j \in \{k+1,\cdots,n\} \setminus H}} \Theta(i, |H|, j) + \sum_{\substack{k+1 \leq j \leq n \\ H \subset \{1,\cdots,n\} \setminus \{j\} \\ i \in \{1,\cdots,k\} \setminus H}} \Theta(j, |H|, i) \tag{6.2}$$

We analyse the first addend and we demonstrate that it is smaller than $1/2$. Since H is every possible subset of $n-2$ indices, then, letting $\ell = |H|$, there are $\binom{n-2}{\ell}$ different

possible sets H. Picking $j \notin H$, we can rewrite the first addend of (6.2) as:

$$\sum_{1\le i\le k} \frac{r_{w_i}}{R} \sum_{0\le \ell\le n-2} \binom{n-2}{\ell} \prod_{1\le h\le \ell} \frac{h\ r_b}{R+(n-h)r_b} \times \\ \times \sum_{k+1\le j\le n,\, j\notin H} \frac{r_{w_j}}{R+(n-1-\ell)r_b} \prod_{1\le a\le 2n-2-\ell} \frac{a\ r_b}{R+a\ r_b} \quad (6.3)$$

First observe that $\frac{a\ r_b}{R+a\ r_b} \le 1$; therefore $\prod_{1\le a\le 2n-2-\ell} \frac{a\ r_b}{R+a\ r_b} \le 1$, as well. Henceforth (6.3) can be over-approximated as

$$\sum_{1\le i\le k} \frac{r_{w_i}}{R} \sum_{0\le \ell\le n-2} \binom{n-2}{\ell} \prod_{1\le h\le \ell} \frac{h\ r_b}{R+(n-h)r_b} \sum_{k+1\le j\le n,\, j\notin H} \frac{r_{w_j}}{R+(n-1-\ell)r_b} \quad (6.4)$$

Moreover, since $\frac{1}{R+(n-h)r_b} \le \frac{1}{(n-h)r_b} \le 1$, then

$$\prod_{1\le h\le \ell} \frac{h\ r_b}{R+(n-h)r_b} \ \le\ \prod_{1\le h\le \ell} \frac{h\ r_b}{(n-h)r_b} \ \le\ \frac{\ell!\ r_b^\ell}{(n-1)\dots(n-\ell)r_b^\ell}\ .$$

Thus, we obtain that (6.4) is less than or equal to:

$$\begin{aligned} &\sum_{1\le i\le k} \frac{r_{w_i}}{R} \sum_{0\le \ell\le n-2} \binom{n-2}{\ell} \frac{\ell!}{(n-1)\dots(n-\ell)} \sum_{k+1\le j\le n,\, j\notin H} \frac{r_{w_j}}{R+(n-1-\ell)r_b} \\ &\le \sum_{1\le i\le k} \frac{r_{w_i}}{R} \sum_{0\le \ell\le n-2} \frac{(n-2)!}{\ell!(n-2-\ell)!} \frac{\ell!}{(n-1)\dots(n-\ell)} \sum_{k+1\le j\le n,\, j\notin H} \frac{r_{w_j}}{R+(n-1-\ell)r_b} \\ &\le \sum_{1\le i\le k} \frac{r_{w_i}}{R} \sum_{0\le \ell\le n-2} \frac{(n-2)\dots(n-1-\ell)}{(n-1)\dots(n-\ell)} \sum_{k+1\le j\le n,\, j\notin H} \frac{r_{w_j}}{R+(n-1-\ell)r_b} \\ &\le \sum_{1\le i\le k} \frac{r_{w_i}}{R} \sum_{0\le \ell\le n-2} \frac{n-1-\ell}{n-1} \sum_{k+1\le j\le n,\, j\notin H} \frac{r_{w_j}}{R+(n-1-\ell)r_b} \end{aligned} \quad (6.5)$$

The sum of r_{w_j} terms with $j \notin H$ can be over-approximated by the sum of all the terms

of the second partition, i.e.

$$\sum_{k+1\leq j\leq n,\, j\notin H} r_{w_j} \leq \sum_{k+1\leq j\leq n} r_{w_j} \ .$$

As a consequence, (6.5) can be over-approximated as follows:

$$\begin{aligned}
&\sum_{1\leq i\leq k} \frac{r_{w_i}}{R} \sum_{0\leq \ell\leq n-2} \frac{n-1-\ell}{(n-1)(R+(n-1-\ell)r_b)} \sum_{k+1\leq j\leq n} r_{w_j} \\
&\quad = \quad \frac{1}{R} \sum_{1\leq i\leq k} r_{w_i} \sum_{k+1\leq j\leq n} r_{w_j} \sum_{0\leq \ell\leq n-2} \frac{n-1-\ell}{(n-1)(R+(n-1-\ell)r_b)} \\
&\quad \leq \quad \frac{1}{R} \sum_{1\leq i\leq k} r_{w_i} \sum_{k+1\leq j\leq n} r_{w_j} \sum_{0\leq \ell\leq n-2} \frac{n-1-\ell}{(n-1)((n-1-\ell)r_b)} \\
&\quad \leq \quad \frac{1}{R\ r_b} \sum_{1\leq i\leq k} r_{w_i} \sum_{k+1\leq j\leq n} r_{w_j} \sum_{0\leq \ell\leq n-2} \frac{n-1-\ell}{(n-1)(n-1-\ell)} \\
&\quad \leq \quad \frac{1}{R\ r_b} \sum_{1\leq i\leq k} r_{w_i} \sum_{k+1\leq j\leq n} r_{w_j} \sum_{0\leq \ell\leq n-2} \frac{1}{n-1} \\
&\quad \leq \quad \frac{1}{R\ r_b} \sum_{1\leq i\leq k} r_{w_i} \sum_{k+1\leq j\leq n} r_{w_j}
\end{aligned}$$

since $\sum_{0\leq \ell\leq n-2} \frac{1}{n-1} \leq \frac{n-1}{n-1} = 1$. Furthermore, we have that

$$\sum_{1\leq i\leq k} r_{w_i} \sum_{k+1\leq j\leq n} r_{w_j} \leq \sum_{1\leq i\leq n} r_{w_i} \sum_{1\leq i\leq n} r_{w_i} = R^2,$$

then it follows that

$$\frac{1}{R\ r_b} \sum_{1\leq i\leq k} r_{w_i} \sum_{k+1\leq j\leq n} r_{w_j} \leq \frac{R^2}{R\ r_b} \leq \frac{R}{r_b} \leq \frac{1}{2} \ .^{1}$$

[1]The inequality $\frac{R}{r_b} \leq \frac{1}{2}$ is verified $\forall\ r_b$ such that $\mathtt{r}_b \geq 2\ R$. In particular, we know that the approximated time for a block to be delivered in a Bitcoin system is 12.6 seconds (see [27]), thus $r_b = 1/12.6$, and we have defined R such that $R = \sum_i r_{w_i} = \sum_i (mR \times hR_i)$. Thus, we can state that, in our analyses, the inequality is verified.

Thus, for every $n > 2$:

$$\sum_{\substack{1\leq i\leq k\\ H\subset\{1,\cdots,n\}\setminus\{i\}\\ j\in\{k+1,\cdots,n\}\setminus H}} \Theta(i,|H|,j) \leq \frac{1}{R\ r_b} \sum_{1\leq i\leq k} r_{w_i} \sum_{k+1\leq j\leq n} r_{w_j} \leq \frac{1}{2}$$

The same approach can be taken when considering the second addend of the equation (6.2). In this case we obtain, for every $n > 2$:

$$\sum_{\substack{k+1\leq j\leq n\\ H\subset\{1,\cdots,n\}\setminus\{j\}\\ i\in\{1,\cdots,k\}\setminus H}} \Theta(j,|H|,i) \leq \frac{1}{R\ r_b} \sum_{1\leq i\leq k} r_{w_i} \sum_{k+1\leq j\leq n} r_{w_j} \leq \frac{1}{2}$$

Therefore, it follows that:

$$\sum_{\substack{1\leq i\leq k\\ H\subset\{1,\cdots,n\}\setminus\{i\}\\ j\in\{k+1,\cdots,n\}\setminus H}} \Theta(i,|H|,j) + \sum_{\substack{k+1\leq j\leq n\\ H\subset\{1,\cdots,n\}\setminus\{j\}\\ i\in\{1,\cdots,k\}\setminus H}} \Theta(j,|H|,i)) \leq \frac{2}{R\ r_b} \sum_{1\leq i\leq k} r_{w_i} \sum_{k+1\leq j\leq n} r_{w_j} \leq 1 \tag{6.6}$$

Part 2: the formula (6.1) **is** $P_{(m+1)_{fork}}$**.** The probability is computed by analysing the states of the transition system in detail. To illustrate the technique we first focus on the simple case that nodes 1 and $k+1$ mine before any forwarding of messages. So assume to be in a state $S_{1,k+1}$ where such minings have already occurred. From $S_{1,k+1}$ it is possible to reach a completed state of fork $m+1$ with computations of length $4n-4$ (because there are two blocks to be delivered and added to $n-1$ nodes). The probability that

one of these computations is chosen is

$$\prod_{1\leq a\leq 2n-2} \frac{r_b}{R + a\ r_b}\ .$$

Furthermore, since the rate of adding the new block to the local ledger is way higher, we can suppose that when a miner receives a new block, it immediately adds it to its ledge; this action is represented by the rate r. The probability of the corresponding transition is

$$\frac{r}{R + (a-1)\ r_b + r}\ .$$

Next, we notice that there are $2(2n-2)!$ different computations that reach a completed state of fork $m+1$. Therefore the upper bound of the whole probability is

$$\prod_{1\leq a\leq 2n-2} \frac{a\ r_b}{R + a\ r_b} \frac{r}{R + (a-1)\ r_b + r} \leq \prod_{1\leq a\leq 2n-2} \frac{a\ r_b}{R + a\ r_b}\ .$$

This is due to the fact that every node eventually will receive the blocks and add them to the local ledger and $r > r_b$. In general, we may write this probability as a function

$$\Psi(u, v) = \prod_{1\leq a\leq 2u-2-v} \frac{a\ r_b}{R + a\ r_b}$$

where u is the number of nodes of the system and v is the number of nodes that have received the first block ($v \leq n-2$ because otherwise we do not have forks). Since the probability to reach $S_{1,k+1}$ from the initial state is

$$\frac{r_{w_1}}{R} \frac{r_{w_{k+1}}}{R + (n-1)r_b}$$

then the probability to reach a completed state of fork $m+1$ starting from the initial

state and traversing $S_{1,k+1}$ is

$$\frac{r_{w_1}}{R} \frac{r_{w_{k+1}}}{R + (n-1)r_b} \Psi(n,0) \ .$$

We notice that there is a symmetric state in the system, where node $k+1$ mines before node 1. The composite probability of these two states is therefore

$$2 \times \frac{r_{w_1}}{R} \frac{r_{w_{k+1}}}{R + (n-1)r_b} \Psi(n,0) \ .$$

We are in place to compute the probability that the two nodes that mine are i and j, where $1 \leq i \leq k$ and $k+1 \leq j \leq n$. This probability is

$$2 \times \sum_{1 \leq i \leq k, k+1 \leq j \leq n} \frac{r_{w_i}}{R} \frac{r_{w_j}}{R + (n-1)r_b} \Psi(n,0) \ .$$

The general case is when

1. a node $i \in \{1, \cdots, k\}$ mines,
2. the new block is communicated to a set of nodes H and
3. a node in $\{k+1, \cdots, n\} \setminus H$ mines.

The probability of this case is $\Theta(i, |H|, j)$:

$$\frac{r_{w_i} \ r_{w_j}}{R \ (R + (n-1-|H|)r_b)} \left(\prod_{1 \leq h \leq |H|} \frac{h \ r_b}{R + (n-h)r_b} \right) \Psi(n, |H|)$$

where the factor

$$\left(\prod_{1 \leq h \leq |H|} \frac{h \ r_b}{R + (n-h)r_b} \right)$$

is the probability of nodes in H to receive and immediately add to the local ledger the first block mined. Henceforth, the probability to reach a perfect state with fork $m+1$

from the initial state is

$$\sum_{\substack{1\leq i\leq k\\ H\subset\{1,\cdots,n\}\setminus\{i\}\\ j\in\{k+1,\cdots,n\}\setminus H}} \Theta(i,|H|,j) + \sum_{\substack{k+1\leq j\leq n\\ H\subset\{1,\cdots,n\}\setminus\{j\}\\ i\in\{1,\cdots,k\}\setminus H}} \Theta(j,|H|,i)$$

which is exactly what stated in the theorem. □

The statement of Theorem 6.1 can be generalised to the most general case by supposing that every miner has its own view of the ledger. In this scenario, the proof is more straightforward compared to Theorem 6.1 because every node may mine after the first one.

Corollary 6.1. Let S be a completed state of a Bitcoin system consisting of n miners having ledger $\mathtt{L}_1 \neq \ldots \neq \mathtt{L}_n$. Then the probability $P_{(m+1)_{fork}}$ to reach a completed state with fork of length $m+1$ is smaller than

$$\sum_{\substack{1\leq i\leq n\\ H\subset\{1,\ldots,n\}\setminus\{i\}\\ j\in\{1,\ldots,n\}\setminus H}} \Theta(i,|H|,j)$$

where $R = \sum_{j=1}^{n} r_{w_j}$ and

$$\Theta(i,\ell,j) = \frac{r_{w_i}\ r_{w_j}}{R\ (R+(n-1-\ell)r_b)} \prod_{1\leq h\leq \ell} \frac{h\ r_b}{R+(n-h)r_b} \prod_{1\leq a\leq 2n-2-\ell} \frac{a\ r_b}{R+a\ r_b}\,.$$

Remark. The same method as Corollary 6.1 can be used to calculate the likelihood that a Bitcoin system, which is currently in a consistent state where all nodes have the same ledger, will deteriorate into an inconsistent state, *i.e.* the probability of a fork of length 1. The upper bound for the probability will be the same, because, also in this case, every miner can mine after the first one.

6.2 Double Spending Attack

In this section, we model and analyse an attack on Bitcoin that has been described in [62], namely a hostile miner tries to create an alternate chain faster than the honest one. This scenario admits that an attacker can be convinced that a transaction has been accepted and then create a new branch of the chain, longer than the valid one, with some other transaction spending the same money (double spending attack).

Let $\texttt{Miner}_{Hack}$ be the dishonest miner; technically, its behaviour differs from $\texttt{Miner}_i$ because it mines on a block $\texttt{b}_{Hack}$ that is not the correct block (e.g. the handle of the ledger). In particular, the code of the $\texttt{Miner}_{Hack}$ is the one presented in Listing 5.2, except for the fact that the newly created block is not the child of the handle of the ledger. This means that the operation

$$\texttt{b}_{Hack} \;=\; \textbf{createB}(\texttt{Miner}_{Hack}, \texttt{c}_i, \texttt{L}_{Hack})$$

returns a block child of an ad hoc block in the ledger.

Following the same pattern of Section 6.1, we obtain a theorem to calculate the probability that the attacker can create an alternate chain faster than the rest of the network. Note that $r_{w_{Hack}} = \texttt{mR}\times\texttt{hR}_{Hack}$ where $\texttt{hR}_{Hack}$ is the hashing power owned by the attacker.

Theorem 6.2. Let S be a completed state of a Bitcoin system of n miners with exactly one that is hostile and let $r_{w_{Hack}}$ its mining rate. The probability $P(S_m)$ to reach a perfect state where the hostile miner has created an alternate chain longer than the honest one from m, $m \geq 1$, blocks behind is smaller than

$$\sum_{k\geq 1}\Big[\Phi(r_{w_{Hack}}, r_b, R)^k \Big(\sum_{1\leq j\leq n-1} \Phi(r_{w_j}, r_b, R)\Big)^{k-1}\Big]^m$$

where $R = \sum_{j=1}^{n} r_{w_j}$ and $\Phi(r_w, r_b, R) = \dfrac{r_w}{R}\prod_{1\leq a\leq n-1}\dfrac{a\; r_b}{R+(n-a)r_b}$.

As for Theorem 6.1, the technique used for demonstrating the above statement consists of analysing the stochastic transition system. To explain the probability, assume

to be in a completed state and compute the probability to reach a completed state in which the dishonest node has created an alternate chain from m blocks behind. We start by computing the probability that the dishonest node $\texttt{Miner}_{Hack}$ has caught up by one block. This kind of attack succeeds if $\texttt{Miner}_{Hack}$ mines one block and this happens with probability $\frac{r_{w_{Hack}}}{R}$. It may also happen that honest nodes mine k blocks and $\texttt{Miner}_{Hack}$ mines $k+1$ blocks in the same amount of time. The formal proof is given below.

Proof. Assume to be in a completed state in which every node has the same ledger, i.e. $\texttt{L}_i = \texttt{L}_j$, $\forall i, j \in \{1, \ldots, n\}$. We want to compute the probability to reach a perfect state in which the dishonest node has created an alternate chain from m blocks behind. We start by computing the probability that the dishonest node $\texttt{Miner}_{Hack}$ has caught up by one block. This kind of attack succeeds if $\texttt{Miner}_{Hack}$ mines one block and all the other nodes receive the block, i.e. the probability is:

$$\frac{r_{w_{Hack}}}{R} \prod_{i=1}^{n-1} \frac{i\ r}{R + (n-i)r}$$

Otherwise, it succeeds also in the case the honest nodes mine k blocks and $\texttt{Miner}_{Hack}$ mines $k+1$ blocks in the same amount of time. Thus, we obtain the formula

$$\sum_{k=1}^{+\infty} \Big(\frac{r_{w_{Hack}}}{R} \prod_{i=1}^{n-1} \frac{i\ r}{R + (n-i)r} \Big)^k \Big(\sum_{j=1}^{n-1} \frac{r_{w_j}}{R} \prod_{i=1}^{n-1} \frac{i\ r}{R + (n-i)r} \Big)^{k-1}$$

Now, it is trivial to prove that the probability that $\texttt{Miner}_{Hack}$ creates an alternative chain one block longer than the original chain from m blocks behind is

$$\Big[\sum_{k=1}^{+\infty} \Big(\frac{r_{w_{Hack}}}{R} \prod_{i=1}^{n-1} \frac{i\ r}{R + (n-i)r} \Big)^k \Big(\sum_{j=1}^{n-1} \frac{r_{w_j}}{R} \prod_{i=1}^{n-1} \frac{i\ r}{R + (n-i)r} \Big)^{k-1} \Big]^m$$

Finally, we have to prove that $P(S_m)$ is less than one. By observing that $\prod_{1 \leq i \leq n-1} \frac{i\ r}{R+(n-i)r} \approx 1$, we can show that $P(S_1) \leq 1$ and then, since $P(S_m) = P(S_1)^m$, we are able to prove

the statement. In particular:

$$
\begin{aligned}
P(S_1) &= \sum_{k\geq 1}\Big(\frac{r_{w_{Hack}}}{R}\prod_{1\leq i\leq n-1}\frac{i\ r}{R+(n-i)r}\Big)^k\Big(\sum_{1\leq j\leq n-1}\frac{r_{w_j}}{R}\prod_{1\leq i\leq n-1}\frac{i\ r}{R+(n-i)r}\Big)^{k-1} \\
&\approx \sum_{k=1}^{+\infty}\Big(\frac{r_{w_{Hack}}}{R}\Big)^k\Big(\sum_{1\leq j\leq n-1}\frac{r_{w_j}}{R}\Big)^{k-1} = \sum_{k\geq 1}\Big(\frac{r_{w_{Hack}}}{R}\Big)^k\Big(1-\frac{r_{w_{Hack}}}{R}\Big)^{k-1} \\
&= \sum_{k\geq 1}\frac{\Big(\frac{r_{w_{Hack}}}{R}\Big)^k\Big(1-\frac{r_{w_{Hack}}}{R}\Big)^k}{1-\frac{r_{w_{Hack}}}{R}} = \sum_{k\geq 1}\frac{\Big(\frac{r_{w_{Hack}}}{R}-\frac{r^2_{w_{Hack}}}{R^2}\Big)^k}{1-\frac{r_{w_{Hack}}}{R}} \\
&= \frac{R}{R-r_{w_{Hack}}}\sum_{k=1}^{+\infty}\Big(\frac{r_{w_{Hack}}}{R}-\frac{r^2_{w_{Hack}}}{R^2}\Big)^k
\end{aligned}
$$

This is a geometric series with common ratio $\rho = \frac{r_{w_{Hack}}}{R} - \frac{r^2_{w_{Hack}}}{R^2} \in \big[0,1)$ because by hypothesis $h_{Hack}, R \in [0,1]$ and $r_{w_{Hack}} = h_{Hack}\ R$. Thus, we have

$$
\begin{aligned}
&\frac{R}{R-r_{w_{Hack}}}\sum_{k=1}^{+\infty}\Big(\frac{r_{w_{Hack}}}{R}-\frac{r^2_{w_{Hack}}}{R^2}\Big)^k = \frac{R}{R-r_{w_{Hack}}}\left(\frac{1}{1-\frac{r_{w_{Hack}}}{R}+\frac{r^2_{w_{Hack}}}{R^2}}-1\right) \\
&= \frac{R}{R-r_{w_{Hack}}}\ \frac{\frac{r_{w_{Hack}}}{R}(1-\frac{r_{w_{Hack}}}{R})}{1-\frac{r_{w_{Hack}}}{R}+\frac{r^2_{w_{Hack}}}{R^2}} = \frac{\frac{r_{w_{Hack}}}{R}}{1-\frac{r_{w_{Hack}}}{R}+\frac{r^2_{w_{Hack}}}{R^2}} \\
&= \frac{r_{w_{Hack}}\ R}{R^2-r_{w_{Hack}}\ R+r^2_{w_{Hack}}} = \frac{h_{Hack}\ R^2}{R^2-h_{Hack}\ R^2+h^2_{Hack}\ R^2} \\
&= \frac{h_{Hack}}{1-h_{Hack}+h^2_{Hack}} \leq 1,\ \forall h_{Hack}\geq 0
\end{aligned}
$$

Since $P(S_1) \leq \frac{r_{w_{Hack}}\ R}{R^2-r_{w_{Hack}}\ R+r^2_{w_{Hack}}} \leq 1$, it immediately follows that

$$
P(S_m) = P(S_1)^m \leq \Big(\frac{r_{w_{Hack}}\ R}{R^2-r_{w_{Hack}}\ R+r^2_{w_{Hack}}}\Big)^m \leq 1\ .
$$

□

It is worth to notice that this technique is different from the one in [62], where Nakamoto assumed *a priori* that the ratio between the blocks mined by the attacker and those mined by the honest miners is the expected value of a Poisson distribution. In particular, we do not assume that miners' behaviour can be described by a certain statistical model, therefore our context is less restrictive. We also notice that Poisson distribution expresses the probability of a certain event occurs in a time period, independently of the time since the last event. Moreover, Nakamoto models the attack counting the number of minings of the attacker in an interval of time, assuming that the probability for success does not change during the experiment. In our case, the probability is computed as the attacker was a standard node and the attacker's mining activity was in competition with the same process of the other nodes.

Chapter 7

The Hybrid Casper Protocol

This chapter presents the PRISM+ model for the Hybrid Casper protocol. The general model is outlined first, followed by a specific model that includes an attacker, which is detailed in Section 7.1.

From Section 7.2, the results obtained from the protocols are presented. To validate the model, a comparison to literature [20, 21] on Hybrid Casper is made in the first step. The resilience of Hybrid Casper to changes in different protocol parameters is analysed in Section 7.3, and the protocol's ability to withstand well-known attacks is studied in Section 7.4.

7.1 Definition of the Models

We model Hybrid Casper in PRISM+ as the parallel composition of n `Validator` modules and the modules `Vote_Manager`, `Network` and `Global`. The architecture of our model is in Figure 7.1.

The module `Vote_Manager` stores the tables containing the votes for each checkpoint and calculates the rewards/penalties at the end of each epoch; the module `Network` implements the broadcast communication mechanism among validators; `Global` is the auxiliary module that computes the length of forks already presented in the previous chapter (see Listing 5.7 for the definition). We note that the management of votes

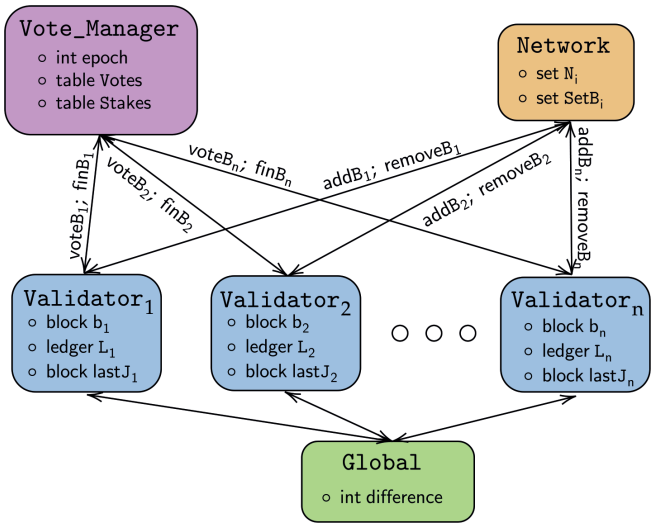


Figure 7.1: The Ethereum PoS model architecture.

is *centralised* in `Vote_Manager`, which corresponds to the smart contract of Hybrid Casper [21].

In our model and in the analyses presented in next sections we overlook some details of Hybrid Casper that are not relevant for the properties we are interested in. In particular, since our goal is to study the behaviour of the protocol to changes of basic parameters, such as the rate of creating new blocks and the percentages in the penalties system, we assume that the network consists of validators that also create new blocks.

For clarity sake, also in this case we present a sugared version of the `PRISM+` code; the online repository[1] contains the actual implementation, the verified properties with the data of our analyses and the instructions for the use of the tool. We present below the `PRISM+` modules for validators, network and vote manager.

The `Validator` process

The code for the `Validator` module is reported in Listing 7.1. A `Validator` is defined as a state machine with seven states. The current state is recorded in the variable $\texttt{STATE}_i$ defined at line 2. The actions of `Validator` are guarded by $\texttt{STATE}_i$ and update this variable when executed.

[1] `https://github.com/adeleveschetti/casper-analysis`

```
module Validator_i
    STATE_i : [Start,Create,Receive,Move,Vote,Check,Fin] init Start;
    b_i : block {Validator_i,0;genesis,0};
    L_i : ledger ⟨{genesis,0;genesis,0}⟩;
    c_i : [0..1000] init 0;
    lastJ_i : block init (genesis^0;genesis^0;0);

    [] (STATE_i=Start) ->
         mR : (b_i'=createB(Validator_i,c_i,L_i))&(c_i'=c_i+1)&(STATE_i'=Create);
    [] (STATE_i=Start) ->lR : (STATE_i'=Receive);
    [] (STATE_i=Start) ->rC : (b_i'=lastCP(L_i))&(STATE_i'=Check);
    [addB_i] (STATE_i=Create)&!isCP(b_i) ->
                1 : (L_i'=addBtoL(L_i,b_i))&(STATE_i'=Start);
    [addB_i] (STATE_i=Create)&isCP(b_i) ->
                1 : (L_i'=addBtoL(L_i,b_i))&(STATE_i'=Vote);
    [voteB_i] (STATE_i=Vote) ->1 : (STATE_i'=Start);
    [] (STATE_i=Receive)&!isEmpty(set_i) ->
                1 : (b_i'=receive(set_i))&(STATE_i'=Move);
    [] (STATE_i=Receive)&isEmpty(set_i) -> 1 : (STATE_i'=Start);
    [removeB_i] (STATE_i=Move)&canBeIns(L_i,b_i)&isCP(b_i) ->
                1 : (L_i'=addBtoL(L_i,b_i))&(STATE_i'=Vote);
    [removeB_i] (STATE_i=Move)&canBeIns(L_i,b_i)&!isCP(b_i) ->
                1 : (L_i'=addBtoL(L_i,b_i))&(STATE_i'=Start);
    [] (STATE_i=Move) & !canBeIns(L_i,b_i) -> 1 : (STATE_i'=Start);
    [] (STATE_i=Check)&isJust(b_i,Votes,Stakes)&lastJ_i=lastboCP(b_i,L_i) ->
                1 : (lastJ_i'=b_i)&(STATE_i'=Fin);
    [] (STATE_i=Check)&isJust(b_i,Votes,Stakes)&lastJ_i!=lastboCP(L_i) ->
                1 : (lastJ_i'=b_i)&(STATE_i'=Start);
    [] (STATE_i=Check)&!isJust(b_i,Votes) ->1 : STATE_i'=Start;
    [finB_i] (STATE_i=Fin) ->1 : STATE_i'=Start;
endmodule
```

Listing 7.1: Pseudocode of a Validator.

In the initial state `Start`, the validator may either create a new block and transit to `Create` state (line 9), or receive a new block from the network and transit to `Receive` (line 10), or check whether a checkpoint can be justified and transit to `Check` (line 11) only if at the end of an epoch.

Since these operations consume time, they are associated with the rates `mR`, `lR`, and `rC`, respectively. The rates `mR` are defined as $1/s$, where s is the number of seconds needed to create a new block, since $s = 14$ in Hybrid Casper [21], then $\texttt{mR} = 1/14$. The rates `lR` are complementary to `mR`, therefore they are defined as $\texttt{lR} = 1 - \texttt{mR}$ (the rational behind this choice is that a validator is more likely not to create a block rather than to create one). The rates `rC` represent how often a validator should check for new justified/finalised blocks. According to [21], this happens at the end of each epoch, thus $\texttt{rC} = 1/(len_{epoch} \times \texttt{s})$.

When the module `Validator` creates a new block, it updates its ledger and sends the newly created block to the process `Network` (see action `addB_i` at lines 12 and 14) to forward it to the other validators. The rate of this action is determined by the companion action in `Network` (i.e. $1 \times r_b$), which expresses the communication latency of the network (*c.f.* line 7 of Listing 7.2). In our model, we set r_b to either 1/12.6, which is the broadcast delay of the Bitcoin network [27] or to 1/7, which is the so-called *(1/2)-network synchrony* – the time to deliver messages is 1/2 of the time to create a block [20]. If the new block is a checkpoint (the height of the block is a multiple of the epoch length), `Validator` transits to `Vote` state, otherwise it returns to `Start` state.

In the state `Vote`, `Validator` votes by synchronising with `Vote_Manager` through the action `voteB_i`; this synchronisation causes the addition of the vote to the table `Votes` (*c.f.* line 16).

When `Validator` tries to receive a new block – state `Receive` – it verifies whether `Network` has blocks to deliver (lines from 17 to 19), and, if it does, the `Validator` transits to the state `Move`; otherwise, it returns to `Start`. In the state `Move`, `Validator` verifies whether the block can be inserted in its own ledger (with the function `canBeIns`, lines from 20 to 23). If this is the case and the block is a checkpoint, `Validator` votes for it, otherwise, returns to the initial state (line 24).

From the initial state `Start`, a validator can also transit to the state `Check` (line 11)

with the rate rC, only when it is at the end of an epoch. In this state, Validator verifies whether the last checkpoint, say C_L, has received the majority of the votes (i.e. C_L has been justified) and whether the last but one checkpoint in the blockchain of C_L, say C_A, is also justified. If this is the case, C_A becomes finalised and C_L becomes justified – in lines 25-26 this is performed by storing C_L in $lastF_i$ and updating the last finalised block of L_i to $lastJ_i$. If C_A is not justified then C_L is stored in $lastJ_i$ (lines 27-28). In any case, Validator goes to Start state and the process starts again.

The Network process

The module Network is defined in Listing 7.2 where the variable N represents the number of validators in the system. For each $Validator_i$, where $1 \leq i \leq N$, the internal state of Network contains (i) the set of blocks set_i that represents the messages to be delivered to $Validator_i$; and (ii) the set N_i that records the nodes to which the $Validator_i$ is connected to.

```
1  module Network
2     for i from 0 to N:
3        set_i : set [];
4        N_i : set [];
5     for i from 0 to N:
6        [addB_i] -> r_b: foreach k in N_i{ set_k'=add(set_k,b_i); }
7     for i from 0 to N:
8        [removeB_i] -> 1 : set_i'=remove(set_i,b_i);
9  endmodule
```

Listing 7.2: Pseudocode of the Network.

In this section, we assume that validators are totally connected, therefore N_i is always equal to $\{1, \cdots, i-1, i+1, \cdots, n\}$. Network module synchronises with $Validator_i$ who creates a block by synchronising on the action addB_i. When this happens, Network updates the sets of blocks of the validators contained in N_i (line 7). When a block has been added to the local ledger of $Validator_i$, by synchronising with the action removeB_i, Network removes the block from the corresponding set (line 8).

The Vote_Manager process

The module Vote_Manager is reported in Listing 7.3. As already explained, the main feature of this process is to store the votes for each checkpoint and calculate the rewards for the validators every time a checkpoint is finalised. The Vote_Manager process includes several fields such as an integer field called epoch that keeps track of the longest epoch in which a checkpoint has been finalised, a map field called stakes that stores the stake of each validator, and a map field called voted that records the validators who have voted for a specific checkpoint.

```
1   module Vote_Manager
2     stakes : map {} ;
3     votes : map {};
4     epoch : [0..1000] init 0;
5
6     for i from 0 to N:
7       stakes[validator_i] : [0..MAX_STAKE] init STAKE_i;
8     for i from 0 to N:
9       [voteB_i] ->1 : votes'=addVote(votes,b_i,Validator_i);
10      [finB_i] (height(lastF(L_i)) > epoch) ->
11        1: epoch'=height(lastF(L_i))&stakes'=updateS(stakes,votes,lastF(L_i));

12      [finB_i] (height(lastF(L_i)) <= epoch) -> 1: ;
13  endmodule
```

Listing 7.3: Pseudocode of the Vote_Manager.

The module synchronises with the i-th validator on actions voteB_i and finB_i. The synchronisation on voteB_i adds the vote for the block that is stored in b_i to the map Votes, i.e. the name $Validator_i$ is added to the list of b_i (line 9). Specifically, the name of the validator who voted is added to the list associated with the block, where the block is the key in the map. The synchronisation on finB_i is used to compute the rewards and penalties for each validator when a block is finalised. In particular, this happens when the first validator finalises a block b because, in this case, the height of b is higher than the value epoch. (This is our modelling of Hybrid Casper's epochs.) It is

worth noticing that the last finalised block is returned by the function `lastF(`L_i`)`. In this case, both `stakes` and `epoch` are updated with the new stakes and `height(b)`, respectively (lines 10–11). If the validator is not the first to finalise then no update occurs.

The Eclipse Attack

In the Eclipse attack, an adversary attempts to obstruct message delivery at the level of the peer-to-peer network causing nodes to work on a corrupted or distorted snapshot of the blockchain [44, 28].

```
1  module Network
2    for i from 0 to N:
3      set_i : set [];
4      N_i : set [];
5      Victims : set [];
6      [comm] ->1 : foreach k in N_i\Victims{ set_k'=add(set_k,b_i); }
7    foreach i in Victims:
8      [addB_i] (attack=true) ->r_b: foreach k in Victims{ set_k'=add(set_k,b_i); }
9    foreach i N_i\Victims:
10     [addB_i] (attack=true) ->r_b: foreach k in N_i\Victims{ set'_k=add(set_k,b_i); }

11   for i from 0 to N:
12     [addB_i] (attack=false) ->r_b: foreach k in N_i{ set_k'=add(set_k,b_i); }
13   for i from 0 to N:
14     [removeB_i] ->1 : set_i'=remove(set_i,b_i);
15 endmodule
```

Listing 7.4: Pseudocode of the Network in presence of an Attacker.

The underlying idea of this attack is that an adversary controls all the incoming and outgoing connections of a victim to prevent it from receiving new blocks from the network. In this way, victims receive new blocks only from the attacker and from other victims. The attacker waits until the blocks created by victims are likely to be justified by the rest of the network. Then, the attacker stops eclipsing and publishes the private

chain to the network. The attack succeeds when a block created by the victims or by the attacker is either justified by the network or the honest validators start using the corrupted chain.

This attack is modelled in PRISM+ as a participant that runs in parallel with honest validators. In particular, we have modified the `Network` code in Listing 7.2 – the new code is in Listing 7.4 – and we use the code in Listing 7.5 for the attacker.

The attacker collects the blocks created by the victims in the set $\texttt{setAtt}_i$. The attacker also counts both the blocks created by the victims and the ones created by the rest of the network, storing these numbers in `nBlocksAttack` and `nBlocks`, respectively. Since the victims are isolated from the rest of the network, they can communicate only between them and with the attacker. As a result, in the Network's code, we have altered the sets of blocks that validators and victims can submit (lines 7-8 of Listing 7.4). As soon as the attacker notices that the blocks created by the victims are more than those created by the rest of the network (line 12 of Listing 7.5), the attacker makes them available to the rest of the network (line 18 of Listing 7.5).

The attack is successful when a block created by the attacker or by one of the victims becomes justified. To define this property, we introduce the boolean variable $\texttt{eclipseAtt}_i$ in every validator; this variables is set to `true` when the validator is about to justify a block created by a victim or by the attacker (lines 30-31). As soon as the attacker publishes all the blocks, the attack is considered over and all the validators can communicate again between each other (lines 11-14 of Listing 7.4).

```
1  module Attacker_i
2    STATE_i : [Start,Create,Receive,Move,Vote,Check,Fin,Comm] init Start;
3    setAtt_i : set [];
4    attack : bool init true;
5    nBlocksAttack : [0..1000] init 0;
6    nBlocks : [0..1000] init 0;
7    eclipseAtt_i : bool init false;
     ...
8    [] (STATE_i=Start) ->
9        mR_i : (nBlocksAttack'=nBlocksAttack+1)&(b_i'=createB(Validator_i,c_i,L_i))
10           &(c_i'=c_i+1)&(STATE_i'=Create);
```

```
11    [] (STATE_i=Start)&(nBlocksAttack>nBlocks)&(!isEmpty(setAtt_i)) ->
12        1 : (attack'=false)&(STATE_i'=Comm)&(b_i'=receive(setAtt_i));
      ...
14    [addB_i] (STATE_i=Create)&!isCP(b_i)&(attack=true) ->
15        1 : setAtt_i'=add(setAtt_i,b_i)&(L_i'=addBtoL(L_i,b_i))&(STATE_i'=Start);
16    [addB_i] (STATE_i=Create)&isCP(b_i)&(attack=true) ->
17        1 : setAtt_i'=add(setAtt_i,b_i)&(L_i'=addBtoL(L_i,b_i))&(STATE_i'=Vote);
      ...
18    [comm] (STATE_i=Comm) ->(setAtt_i'=remove(setAtt_i,b_i))&(STATE_i'=Start);
19    [removeB_i] (STATE_i=Move)&canBeIns(L_i,b_i)&isCP(b_i)&fromVictim(b_i) ->
20        1 : (nBlocksAttack'=nBlocksAttack+1)&(setAtt_i'=add(setAtt_i,b_i))
21           &(L_i'=addBtoL(L_i,b_i))&(STATE_i'=Vote);
22    [removeB_i] (STATE_i=Move)&canBeIns(L_i,b_i)&!isCP(b_i)&fromVictim(b_i) ->
23        1 : (nBlocksAttack'=nBlocksAttack+1)&(setAtt_i'=add(setAtt_i,b_i))
24           &(L_i'=addBtoL(L_i,b_i))&(STATE_i'=Start);
25    [removeB_i] (STATE_i=Move)&canBeIns(L_i,b_i)&isCP(b_i)&!fromVictim(b_i) ->
26        1 : (nBlocks'=nBlocks+1)&(L_i'=addBtoL(L_i,b_i))&(STATE_i'=Vote);
27    [removeB_i] (STATE_i=Move)&canBeIns(L_i,b_i)&!isCP(b_i)&!fromVictim(b_i) ->
28        1 : (nBlocks'=nBlocks+1)&(L_i'=addBtoL(L_i,b_i))&(STATE_i'=Start);
      ...
29    [] (STATE_i=Check)&isJust(b_i,Votes,Stakes)&lastJ_i=lastboCP(b_i,L_i)
30           &fromVictim(b_i) ->
31        1 : (lastJ_i'=b_i)&(STATE_i'=Fin)&(eclipseAtt_i'=true);
32  endmodule
```

Listing 7.5: Pseudocode of an Attacker.

7.2 Coherence of the Model

In this section we report the analyses assessing the coherence of the model. It's worth noting that the systems we are studying have a fixed number of validators, specifically $n = 6, 8, 10, 12, 14, 16$ and we run the experiments until we observe stability in the results.

Usually with networks larger than 10-12 validators, the differences between the outputs of the analyses are in the order of 10^{-3}. This is due to the fact that we use mean rates to describe the latency of the network (which are taken from the literature) therefore the number of nodes has little impact on the broadcast of blocks. In particular, we strongly believe that our conclusions obtained with, say 14-16 validators, are meaningful also for larger networks. Therefore, the following analyses have been carried out by considering 14 validators, because it was a valid trade off between the error percentage and the time needed by `PRISM+` for the analyses.

Before starting analysing different settings for our model, we want to validate it by comparing our results with the one presented in the literature. For this purpose, we set the basic parameters of the model as follows:

(i) $len_{epoch} = 64$, *i.e.* checkpoints are validated every 64 blocks;

(ii) all validators start with the same amount of stake in the initial state and work honestly, *i.e.* they never vote maliciously nor do they create blocks in the wrong position;

(iii) the rate `mR` of creating new blocks is 1/14.

Our model may be easily updated to analyse validators (or pools of validators) with different mining rates: it suffices to set the constants `mR` to the corresponding values. Since there are not many works on the Hybrid Casper protocol and there are not real values and parameters because it has been used to guaranteed a smooth transition between the proof of work and the proof of stake Ethereum protocol, we compare our results with the ones showed in [20] where the authors present the protocol.

We start by computing the probabilities for creating a new block. Figure 7.2 reports these probabilities, which are calculated by letting `PRISM+` to analyse the property:

```
P=?[F<=T "someCreated"]
```

where `someCreated` is a label that identifies all the states in which a validator is in the state `Create`. In Figure 7.2 and in Figure 7.3, the broadcast delay r_b is set to 1/7. The reason behind this choice is the fact that in [20] the authors analyse the protocol with

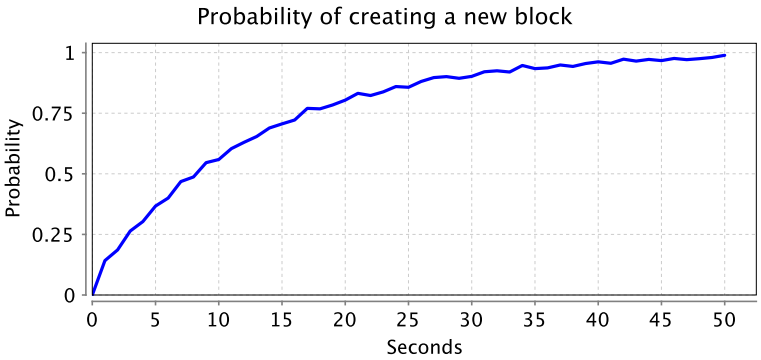


Figure 7.2: Probability of creating a block.

the assumption that the time to deliver messages is 1/2 of the time to create a block, *c.f. (1/2)-network synchrony*.

The results we obtain are coherent with the ones in the literature [20]; in particular, the probability of creating a block within 14 seconds is 0.6 and the one of creating a block within 50 seconds is 1. To verify the occurrence of forks in ledgers, we use the module `Global` already defined in Chapter 5.2 (Listing 5.7). The only difference between the two modules is the fact that the one used in the Hybrid Casper model computes the difference between the ledgers of the system every time the $\texttt{Validator}_i$ changes its state to `Start`, instead of when the node goes into the `Add` state. Therefore, the probability of reaching a state with a fork of length k within the first `T` time units is defined by the same exact formula.

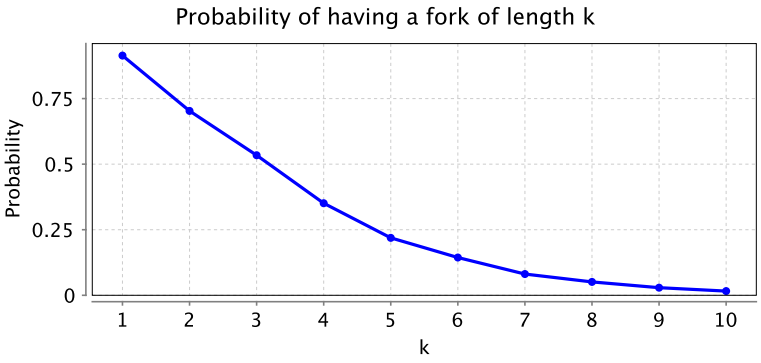


Figure 7.3: Probability of having a fork of length k.

Figure 7.3 shows how the probability of having a fork of length k, with $1 \leq k \leq 10$,

varies over the time. We run the analysis by considering $k*14$ as bound time. Our results show that the probability of a fork of length 1 is higher than 0.9, while the probability decreases as the k increases, and it is 0.009 for forks of length 10.

Figures 7.4 and 7.5 report, respectively, the probabilities of justification and finalisation within k epochs. These probabilities are computed by PRISM+ using the formulas

```
P=?[F<=T "someJustified"]     P=?[F<=T "someFinalised"]
```

where `someJustified` and `someFinalised` are the labels that identify all the states in which a validator is in the state `Check` and the block is justified and finalised, respectively (lines 25-28 of Listings 7.1).

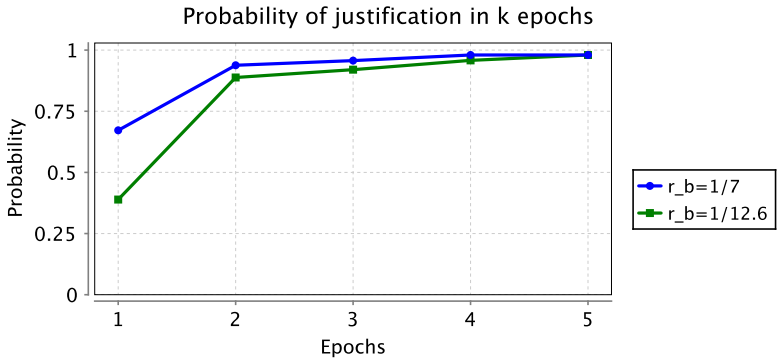


Figure 7.4: Probability of justification within k epochs.

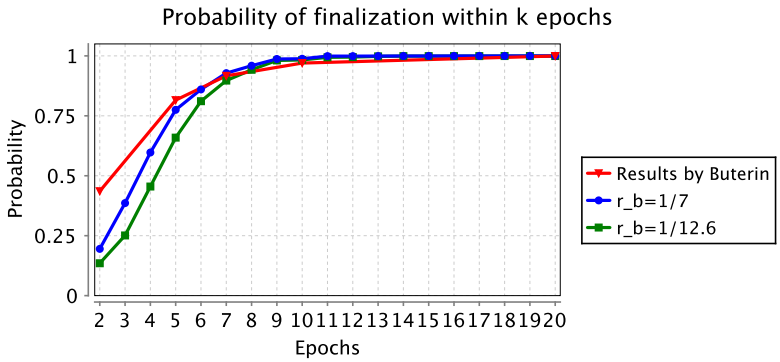


Figure 7.5: Probability of finalisation within k epochs.

The experiments reported in Figures 7.4 and 7.5 have been run with two different broadcast delays: the standard broadcast delay of Bitcoin, *i.e.* $r_b = 1/12.6$ ([27]) and

the (1/2)-network synchrony delay $r_b = 1/7$.

It turns out that, when $r_b = 1/12.6$, the probability of justifying within 1 epoch is 0.389 while it is 0.672 when $r_b = 1/7$. This is because, in the first case, a checkpoint needs more time to reach all the nodes in the network. Therefore, as the voting process becomes longer, the probability when $r_b = 1/12.6$ becomes smaller. It is also worth to notice that, when the epochs are 5, the probability is greater than 0.96 with both values of r_b, which is in accordance with [20] where this probability is stated to be greater than 0.5 in one epoch. We finally notice that the probability of justifying a block is lower when r_b is equal to the rate of Bitcoin.

In Figure 7.5 we report the probability of finalisation within k epochs that have been computed in [20] (the red curve in the figure) and we compare these results with those computed by PRISM+ with the two broadcast values of $r_b = 1/12.6$ and $r_b = 1/7$. Our results are coherent with the literature but a little lower for $k < 7$. We also notice that the probabilities obtained with rates $r_b = 1/12.6$ and $r_b = 1/7$ grows in a similar way and are almost the same for $k > 7$ (for example, when $k = 20$, the probability is 0.9994 with $r_b = 1/12.6$, while it is 0.99999 with $r_b = 1/7$).

Finally, in [20] the authors proved properties of safety and liveness for Hybrid Casper. In particular, they define a protocol to be

- *safe* when two (or more) conflicting finalised checkpoints cannot occur;
- *live* when the set of finalised blocks always grows.

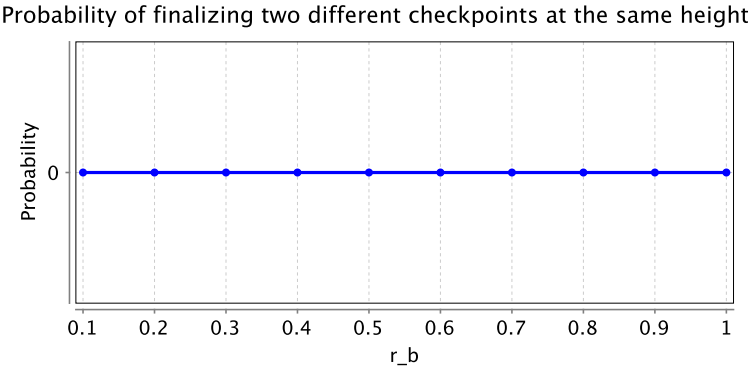


Figure 7.6: Safety property.

Figures 7.6 and 7.7 show the analyses of safety and liveness in PRISM+ with respect to the broadcast delay r_b. In these analyses we embrace the setting adopted in [20], and we suppose that all the validators vote correctly, i.e. they vote only for one checkpoint at the same height. According to our results, the probability of finalising two conflicting checkpoints is always 0 (Figure 7.6) and the probability of finalising a new checkpoint is always greater than 0.85 when $r_b \geq 1/100$ (it is almost 1 with $r_b \geq 1/12.6$).

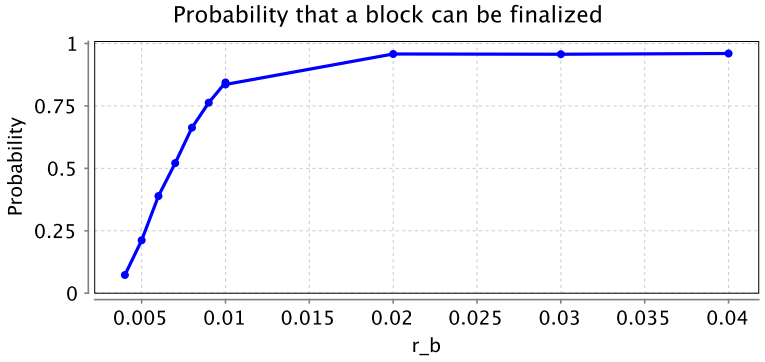


Figure 7.7: Liveness property.

7.3 Hybrid Casper Stress Tests

The resilience of Hybrid Casper to the changes of different parameters of the protocol is verified in this section.

First, we analyse how the probabilities of forks and justifications change by varying the rate mR of creating new blocks. In the following experiments, the broadcast rate r_b is set to 1/7 (we assume it depends on technological constraints of the network; therefore we adhere to the (1/2)-network synchrony assumption). Next, we analyse how the percentage of penalties may affect the behaviour of malicious validators. Hereafter, we will assume that all the mR are equal and we generically denote them with mR.

Different rates of creating blocks. We study the resilience of the protocol when the time for creating blocks is lower than the standard one (14 seconds in Ethereum). More precisely, we consider the cases where the rates are $\mathtt{mR} = 1/14, 1/8, 1/7, 1/6$, respectively,

i.e. blocks are created within 14, 8, 7 and 6 seconds, and the length of epochs is 64.

The results in Figure 7.8 show that the probability of a fork of length k for $\texttt{mR} = 1/8, 1/7, 1/6$ is higher than the one for $\texttt{mR} = 1/14$ (we run the analyses by considering `k*1/mR` as bound time).

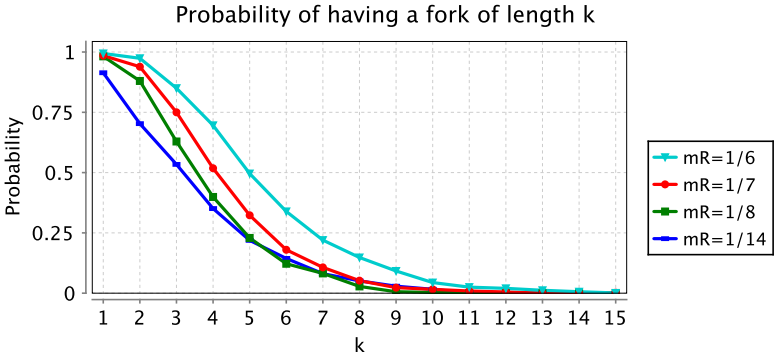


Figure 7.8: Probability of a fork of length k with $\texttt{mR} = 1/4, 1/8, 1/7, 1/6$.

In particular, the probability of a fork of length 7, i.e., $P_{7_{fork}}$, is 0.25 for $\texttt{mR} = 1/6$; whereas it is around 0.06 when we consider the other rates. This may result in a lower probability of justifying a block, because there is a higher probability of having two different checkpoints at the same height. As expected, when $\texttt{mR} = 1/8, 1/7, 1/6$, a block must wait for more epochs to be justified, see Figure 7.10. In particular, with a creation rate $\texttt{mR} \leq 1/7$, a block cannot be justified within one epoch, but after two epochs, the probability rapidly increases. Even when $\texttt{mR} \leq 1/8$ the probability of justifying a block within 1 epoch is lower since the analysis shows that $P_{1\_just} = 0.134$.

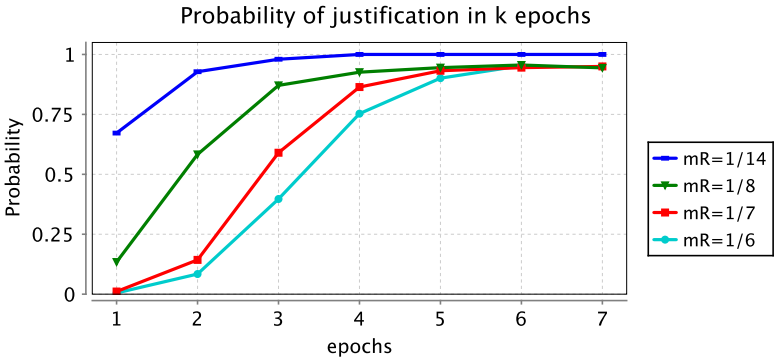


Figure 7.9: Probability of justification within k epochs.

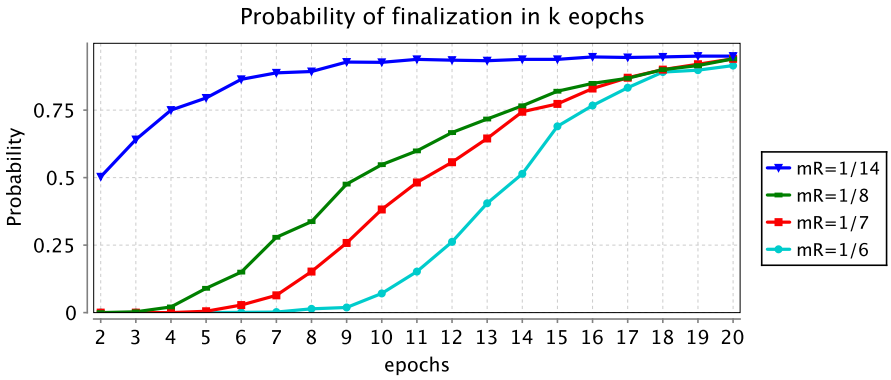


Figure 7.10: Probability of finalisation within k epochs.

Figure 7.10 reports the probability P_{k_fin} of finalisation within k epochs. The results are in line with what we expect, in fact, the slower is the system to create a block the lower is the probability of finalisation within 2 epochs. This is due to the fact that also the probability of justification is very low. Moreover, with $\texttt{mR} = 1/6$, P_{k_fin} starts to increase after 9 epochs, becoming 0.915 after 20 epochs.

Variation of the penalties. We analyse how the stakes change when the penalties for misbehaving validators vary. In Hybrid Casper, to incentivate correct behaviours, a bonus is given to validators when they vote checkpoints that are finalised; on the contrary a penalty is given to those that clearly misbehave.

In the following analyses, we studying how the stakes of malicious validators changes. A validator is malicious when it votes for more than a checkpoint at the same height. To illustrate the technique, we define a stochastic process that misbehaves with a rate of 1/2. (Other strategies can be easily implemented by changing the code.) In particular, the stake of validator j at epoch i is defined by $stake_j(i)$

$$stake_j(0) \stackrel{\text{def}}{=} 10\,ETH$$

$$stake_j(i+1) \stackrel{\text{def}}{=} \begin{cases} stake_j(i) + \gamma \cdot stake_j(i) & \text{if the validator votes correctly} \\ stake_j(i) - \gamma \cdot stake_j(i) & \text{otherwise} \end{cases}$$

that increases or decreases stakes according to a parameter γ. In our experiments (Figure

7.11 and Figure 7.12) we consider three values for γ: 20%, 30% and 40%.

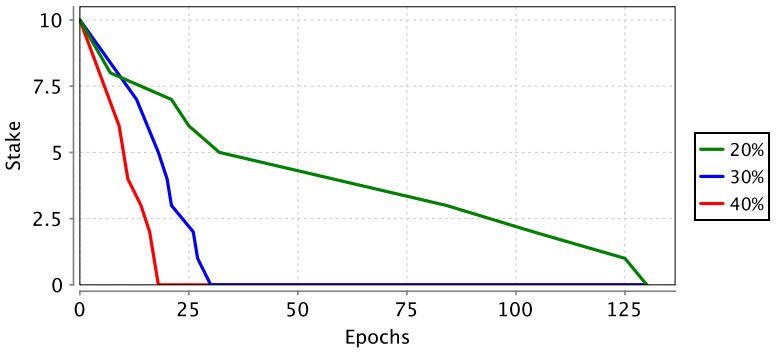


Figure 7.11: How the stake of the malicious validator varies while epochs increase.

In Figure 7.11, we report how the stake of the *malicious* validator varies while epochs increase. It points out that the less the penalty is in percentage, the slower the stake decreases (there are necessary more than 125 epochs to become 0 when γ is 20%).

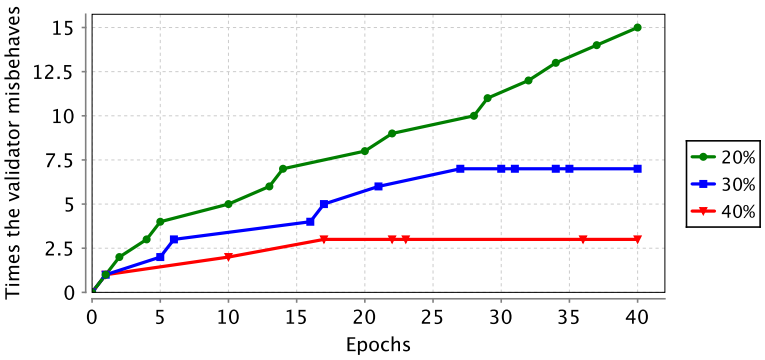


Figure 7.12: The number of times a validator misbehaves with respect to the epochs.

The number of times a validator misbehaves with respect to the epochs is reported in Figure 7.12: the higher the penalty, the lower the number. In particular, when $\gamma = 20$ the validator misbehaves 8 times after 20 epochs, while when $\gamma = 30$ only 6 times. When $\gamma = 40$ the validator misbehaves 3 times at 20 epochs and this number does not grow anymore since its stake becomes 0. This confirms that the malicious behaviour is discouraged when the penalty is high.

We remark that in our model it is always possible to reach a state of fork due to multiple blocks being mined at the same time. Thus, if a honest validator v that follows

the fork choice rule, votes for a block b that will not be voted by the majority (simply because v received b first), v's stake will be shrunk. To illustrate this, we report in Figure 7.13 the updates of the stake of a validator (since we want to highlight the ongoing behaviour of a validator, we have chosen the simulator option in PRISM+). As

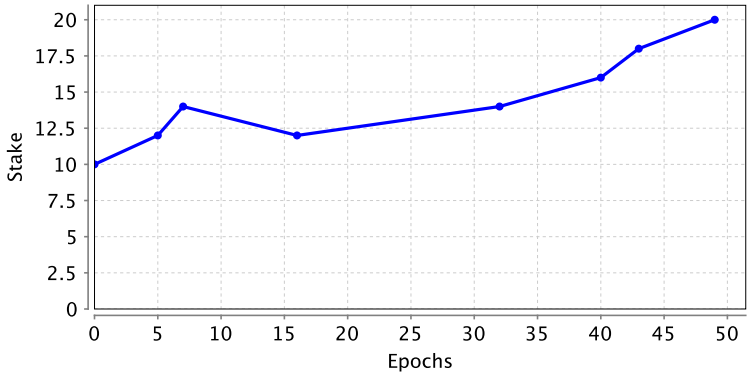


Figure 7.13: How changes the stake of a validator.

the reader can observe, at epoch 16 the stake of the validator decreases, that is due to the fact that it voted for the wrong checkpoint. The same stake increases because the validator voted correctly afterwards.

7.4 Attacks

The purpose of this section is to study the robustness of Hybrid Casper to two attacks that have been studied in the literature [28, 45]. It is worth noticing that all these analyses have been done with little efforts (by manually changing the PRISM+ settings of the protocol), which is a benefit of our technique.

Eclipse Attack. In the Eclipse attack, an adversary attempts to obstruct message delivery at the level of the peer-to-peer network causing nodes to work on a corrupted or distorted snapshot of the blockchain [44, 28]. This attack is modelled in PRISM+ as a participant that runs in parallel with honest validators, the code is presented in Listing 7.5. The honest validators may reach a consensus on this new chain after checking its

validity. The probability of a successful attack by one attacker and an increasing number of victims (from 2 to 5) is computed by the formula

$$\mathbf{P}\texttt{=?[}\mathbf{F}\texttt{<=T "eclipseAttack"]}$$

where `eclipseAttack` means that at least one of the miners' boolean variables $\texttt{eclipseAtt}_i$ is true.

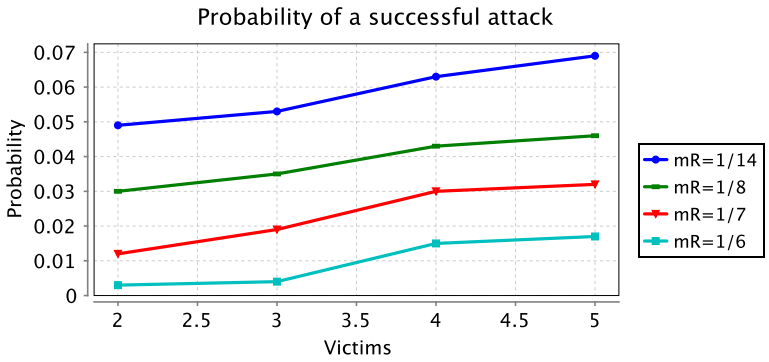


Figure 7.14: The probability of a successful attack by varying the rate of creating new blocks.

Figure 7.14 shows the probability of a successful attack by varying the rate of creating new blocks. In every case, this probability increases with the number of victims. Additionally, when blocks are created faster than the standard ones, then the probability

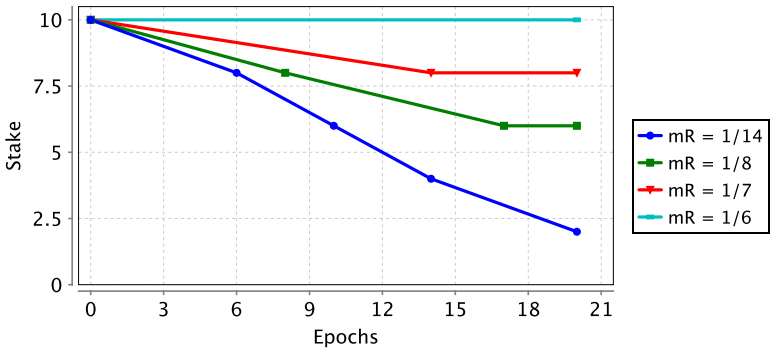


Figure 7.15: How the stake of the victims decreases with respect the number of epochs increases.

of a successful attack is much lower. In particular, when $\texttt{mR} \leq 1/8$, the system appears to be more resilient to this attack.

In a simpler version of this attack, the attacker aims to reduce the stake of the victims. Figure 7.15 reports how the stake of the victims decreases as the length of the epochs changes for different values of mR. It turns out that these results are in line with those of Figure 7.14, that is the attacker does not succeed when $\mathtt{mR} \leq 1/8$.

Majority Attack. In a PoS system a majority attack consists of one validator or a coordinated set of validators that own more than 34% of the overall stake. When this is the case, a majority attack can impact the blockchain by performing finalisation and justification, since checkpoints can now receive the majority of the votes.

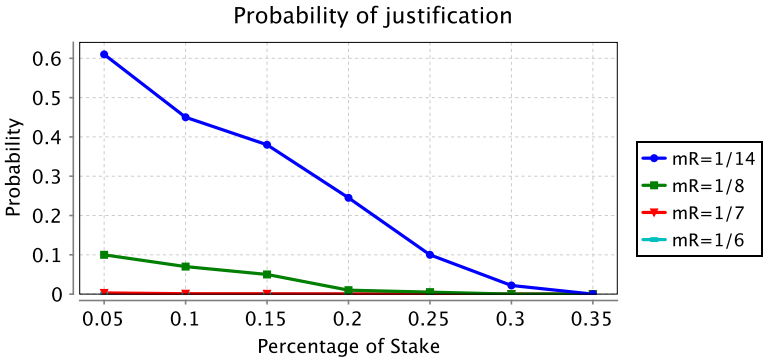


Figure 7.16: Analysis in presence of a majority attack.

Figure 7.16 displays the results obtained by our analysis of the majority attack by changing the total stake owned by the attacker and the rate of creating new blocks. In this case, the code for the attacker is the one reported in Listing 7.1 with the unique difference being the percentage of the stake owned by the attacker.

Our results show that the probability of justifying a block decreases when the percentage of the stake owned by the attacker increases. This is due to the fact that, if the attackers decide to perform a majority attack, the honest part of the network cannot justify (and finalise) new blocks and, thus, the whole system is affected. Additionally, the probability decreases with respect to the rate of creating a new block. This is in line with the idea that the less time is needed to create a block, the faster is the attacker. However, due to the slashing conditions, the majority attack is so costly that it is unlikely that one will ever be launched against this protocol in practice. Thus, in the light of

our results above and considering how fast the stake of a malicious validator decreases, it should not be a problem if one decides to use a faster rate to create new blocks.

7.5 Comparison with Bitcoin

In this section we do a summary to compare the results obtained with the two different protocols with PRISM+ and the ones obtained using Theorem 6.1 presented in Chapter 6 for the Bitcoin protocol.

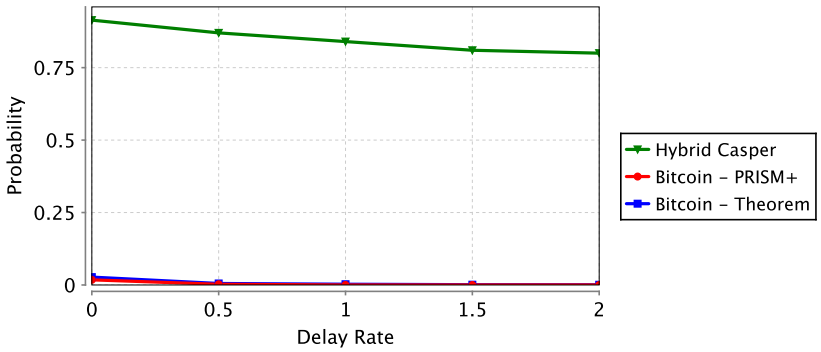


Figure 7.17: Probability of having a fork of length 1 by varying the delay.

In Figure 7.17, we compare the probability of having a fork of length 1 by varying the delay of the network. The probability simulated with the model in PRISM+ and the one calculated using the Theorem 6.1 are very close to each other and, as expected, the theorem gives an upper bound for the probability. For the Hybrid Casper protocol, the probability is way higher and this depends on the fact that the time needed to create a new block is way less. In fact, since a block is created every 14 seconds, it is more likely that two or more nodes create a block at the same time.

Figure 7.18 shows the results of a comparison between the two different blockchain protocols with respect to the probability of forking. In particular, the figure displays the probability of reaching a fork of a certain length k in the presence of a network delay of 14.6 seconds. Also in this case it is worth noticing that the values obtained using Theorem 6.1 are a precise upper bound for the probability of forking in the case of the Bitcoin protocol, as confirmed by PRISM+ simulations. As expected, the Hybrid

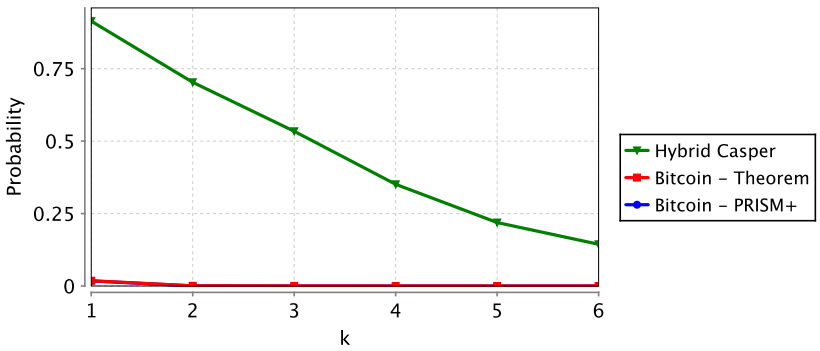


Figure 7.18: Probability of having a fork of length k by varying the delay.

Casper protocol has a higher probability of forking compared to Bitcoin. However, it is important to note that Hybrid Casper includes additional mechanisms, such as finalisation and justification processes, that ensure the security of the protocol despite its higher likelihood of forking. These mechanisms are designed to prevent malicious actors from manipulating the blockchain, even in the presence of forks.

Chapter 8

Related Works

The purpose of this section is to present some of the existing techniques and the state-of-the-art for the formal methods and blockchain area and the main approaches used in literature. In this chapter, we report a comprehensive analysis of all the formal method techniques that have been used to secure blockchain systems. The chapter is organised in three main parts: in Section 8.1 we discuss the broadly classified formal techniques and how they are used in blockchains. Then, we go more into the details of the state of the art concerning the proof of work and proof of stake protocols. In particular, in Section 8.2 we present the works done for the formal analysis of the proof of work and Bitcoin protocol while Section 8.3 focuses on the presentation of the literature about the Hybrid Casper protocol and, more in general, on the proof of stake.

8.1 Formal Methods Applied in Blockchain

Blockchain technology has many implications as a decentralised and distributed framework for maintaining and securing a shared ledger. With the increasing use of blockchain in different fields, formalising and verifying its safety and security features is becoming crucial. Formal methods, which include systematic procedures and formal approaches for system development, can result in software that is bug-free, has minimal defects, is accurate, and is guaranteed to be correct. Due to this, many researchers have decided to

use formal methods to model and analyse blockchain systems. In the following sections, we will discuss the various formal techniques that have been used to analyse blockchain systems.

Process Algebra

Process Algebras are mathematically rigorous languages with well defined semantics that permit describing and verifying properties of concurrent communicating systems. They can be seen as models of processes, regarded as agents that act and interact continuously with other similar agents and with their common environment.

In [48], the authors present the first outcome of Ouroboros [49] formalisation effort: the ♮-calculus (pronounced "natural calculus"), which is a process calculus that they use both as specification and implementation language. They present the language and its semantics. The latter is unique in that it uses a stack of two labeled transition systems to treat phenomena like data transfer and the opening and closing of channel scope in a modular fashion. Since the operational semantics of the ♮-calculus is defined using two transition systems, they have developed an abstract theory of transition systems to treat concepts like bisimilarity.

Communicating sequential processes (CSP) is a formal language that is used to describe patterns of interaction in concurrent systems, it is used to model the behaviour of blockchain systems in [2] by using analytical methods to assess system availability against malicious miner denial-of-service attacks. The research also uses queuing theory to determine the average waiting time for client blockchain transactions when malicious miners are slowing the system down. CSP and queuing theory are used together to test the blockchain's ability to make progress even when malicious miners are present.

Transition System

A transition system is a notion in theoretical computer science that is used to investigate computation. It's a term used to explain how separate systems might behave. It is made up of states and transitions between states that can be labelled with labels from a list; the same label can appear on many transitions. There are various state-transition techniques

and in this section we will discuss how they are used to model and secure blockchain systems.

Petri Nets. A novel approach, based on a Petri Net model to parse the blockchain is presented in [71]. Petri Net is an oriented graph, made of two types of nodes, place and transitions, where each node can be connected only with a node of the other type. Thus, they modelled the Bitcoin blockchain as an oriented graph, made of two types of nodes, addresses and transactions. Their purpose was to define a single useful model in which all main informations about transactions and addresses are represented. Collecting the first 180 thousand blocks, they were able to associate a place for each address and a transition for each Bitcoin transaction. Their Petri net includes pre and post-incidence matrices where all links between addresses and transactions are modelled. The model allowed also to identify a set of behaviours typical of Bitcoin owners, like that of using an address only once, and to reconstruct chains for this behaviour together with the rate of firing.

A similar technique is used in [77], where the authors modelled the most common attacks on blockchains using Petri nets. Their main goal, however, was to analyse the vulnerabilities of blockchains and how quantum computing will affect these or other new attacks in the future.

Markov Decision Processes. Markov decision processes (MDP) model decision making in discrete, stochastic, sequential environments. The essence of the model is that a decision maker, or agent, inhabits an environment, which changes state randomly in response to action choices made by the decision maker. So, they provide a mathematical framework for modelling decision making.

In [79], the authors applied the Markov decision processes to find the optimal selfsh-mining strategy, in which four actions: adopt, override, match and wait, are introduced in order to control the state transitions of the Markov decision process.

The optimal adversarial strategies for double-spending and selfish mining attacks are modelled through MDP in [40]. They also take into account real world constraints such as network propagation, different block sizes, block generation intervals, information propagation mechanism, and the impact of eclipse attacks. With their technique they

are able to show that selfish mining is not always a rational strategy and that the higher the block reward of a blockchain the more resilient it is against double-spending, as well as other results concerning how the block size impacts security and how block rewards and confirmation time are related.

A similar approach is taken in [75], where the goal in this paper is to better understand the conditions under which Bitcoin is resilient to selfish mining attacks. They provide an efficient algorithm that computes an optimal selfish mining policy and they evaluate different protocol modifications that were suggested as countermeasures for selfish mining. They show that in a model that accounts for the delay of block propagation in the network, attackers of any size can profit from selfish mining. They also discuss the interaction between selfish mining attacks and double spending attacks. The work presented in [79] uses these techniques to analyse optimal double spending strategies. In particular, they present a variety of different interpretations of the security of a single transaction in the Bitcoin system, and matching advice regarding the number of confirmations merchants should await in order to properly secure their transactions.

A framework of block-structured Markov processes for blockchain systems is the main contribution of [56]. They are also able to provide an effective method for computing the average transaction–confirmation time of any transaction in a more general blockchain system.

8.2 Proof of Work Blockchains

The blockchain protocol was introduced by Haber and Stornetta [43] and only in the last few years, because of Bitcoin, the problem of analysing the consistency of the ledgers has attracted the interest of several researchers. The discussion of the mainstream blockchain consensus algorithms and the way the classic Byzantine consensus can be revisited for the blockchain context is presented in a paper [42], where the Bitcoin and Ethereum consensus algorithms are described and the behaviour of each process involved in the system is illustrated through pseudo-code. They discuss proof-of-work consensus and illustrate the differences between the Bitcoin and the Ethereum proof-of-work consensus algorithms. Based on these definitions, they warn about the dangers of using these

blockchains without understanding precisely the guarantees their consensus algorithm offers. In particular, they survey attacks against the Bitcoin and the Ethereum consensus algorithms. Then, they also discuss the advantage of the recent Blockchain Byzantine consensus definition over previous definitions, and the promises offered by emerging consistent blockchains. In [39] the authors prove the correctness of the protocol when the network communications are synchronous and focusing on *persistence* and *liveness*. Persistence states that once a transaction goes more than k blocks "deep" into the blockchain of one honest player, then it will be included in every honest player's blockchain with overwhelming probability, and it will be assigned a permanent position in the ledger. On the other hand, liveness says that all transactions originating from honest account holders will eventually end up at a depth more than k blocks in an honest player's blockchain, and hence the adversary cannot perform a selective denial of service attack against honest account holders. For both the properties, the authors require an honest majority of nodes. The extension of this analysis to dynamic asynchronous networks with bounded delays can be found in [68]. There, the authors also provide an abstract model of the Bitcoin protocol that ignores all irrelevant implementation details. The abstract model enables them to formally study the behaviour of the protocol and to detect where there is room for improvement.

The authors in [73], instead, propose a formalisation of blockchain consensus with a proof of its consistency mechanised in a proof assistant. They present an operational model that provides an executable semantics of the system and prove a form of eventual consistency focusing on the notion of global system safety.

Similarly to these papers, we propose an abstract model of Bitcoin where we ignore all the implementation details which do not affect the properties of interest. The main difference between these contributions and our work is that we formalise the blockchain protocol as a stochastic system (with exponential distribution of duration) and prove its properties by using the `PRISM+` model checker.

An empirical analysis of the announcement and spread of blocks that result in the forking of the Bitcoin blockchain is conducted in [65]. The authors examine the time differences in the publication of competing blocks and found that the delay in block propagation among miners is similar to the delay experienced by the average Bitcoin

user. Moreover, they show that the longer a block has been published before a competing block appears, the higher the probability that it will become part of the main chain. The authors also observe a high frequency of short block intervals between two consecutive blocks mined by the same miner after a fork, which could be due to selfish mining or other causes. Finally, they note that the propagation speeds of competing blocks vary greatly, in addition to the time difference in their publication. The authors operated two monitor nodes that establish connections to all reachable peers of the Bitcoin P2P network so they have been able to observe reale measurements of the Bitcoin network, on the contrary we model the Bitcoin protocol as a parallel composition of processes and we use model checking techniques to analyse those properties.

A theoretical approach is presented in [78], where the authors provide a theoretical model for performance modelling and analysis of the Bitcoin network using an Erdös-Rényi random graph model. In particular, they investigate the cause and length of forks for the Bitcoin network considering affecting variables such as block propagation delay, network bandwidth, and block size. They show that the impact of how nodes are connected in the network is very minimal in terms of blockchain forks, on the contrary this value is affected in a significant way by the network delay and the fork probability increases proportionally to the block size. The technique used in this article differs from our approach, however the results are in line with the analyses presented in Chapter 5.

There are few works in the literature that have followed a research line similar to ours, i.e. studying the properties of the Bitcoin protocol using a probabilistic model checker. The articles [30, 22] use UPPAAL [11, 26]. The former studies the security of the proof of work consensus when the network has an adversary miner that leverages the selfish mining strategy introduced in [33]. In particular, their experiments show the effectiveness of selfish mining against various deployment parameters. They show how to model the protocol using the Statistical Model Checking UPPAAL, and identify concrete security properties of the protocol. They also use the model to demonstrate how design decisions can impact different concrete backbone protocol properties in different ways, in a manner that is not obvious from prior asymptotic analysis and it is possible only by doing model checking.

The authors in [22] analyse the probability of success of a double spending attack in

the Bitcoin protocol. They show that double spending can be achieved if the parties in the Bitcoin protocol behave maliciously. In these two works, the main goal is to verify the resilience of Bitcoin by analysing the probability of a successful attack. In contrast, we also study the properties of the protocol under different circumstances and settings. Another difference is that the foregoing works do not model churning nodes and only consider the case in which a block is broadcast to all the other nodes in the system. In our analyses, we also modelled different kinds of topologies and, thus, we cover a more general case.

The work presented in [10] uses PRISM to analyse the so-called 51%-attacks (a pool can attack the network as soon as it reaches a substantial percentage of hash power) in an extension of the Bitcoin protocol (the two phase proof of work). The author proves that the extension of Bitcoin is effective at preventing the 51%-attacks. As in our work, each miner is modelled as a module in the PRISM language; however the work focuses on the actual cryptographic problem and does not implement blocks and the blockchain data structures. In particular, in our study, every cryptographic detail that does not affect our analysis is ignored, but we analyse what actually happens inside the blockchains. This is possible because we extended the model checker PRISM.

Some papers propose stochastic models to analyse specific parts of blockchain systems. In [12], the authors focus on miners and propose a game-theoretic approach to analyse the strategies miners can adopt and the kind of equilibrium these strategies can lead to in blockchain dynamics. Overall, the paper argues that the blockchain protocol is essentially a coordination game with several potential outcomes. There are multiple equilibria, some of which involve forks that cause orphaned blocks and sustained divergence between chains. Additionally, the authors show that the occurrence of forks can be triggered by factors such as delays in information processing and updates to the software.

A similar approach is adopted by in [86] which proposes a formal mathematical framework, to model the core concepts in blockchain-enabled economies. The authors illustrate the dynamics of the blockchain economies simulating and testing two different block reward strategies. The main difference with respect to our work is that their analyses focus on economic aspects. In fact, we do not model miners' strategies and we do not consider the rewards obtained by the miners when they mine a valid block. Their

main goal is to understand what the economic forces at the root of forks are; our analysis instead focuses on the security and integrity aspects of the system.

Moreover, the authors [72] propose a basic stochastic model for the blockchain protocol to capture the block creation and broadcasting process. They model blocks as abstract objects with just the necessary information to analyse the ledger evolution. They also propose a framework to ease the tuning of the model and exploit Monte Carlo simulations to obtain probabilistic results on consistency of the ledgers. In contrast with our purpose, their main goal is to check the ability to detect and prevent double-spending attacks of blockchain protocols.

8.3 Proof of Stake Blockchains

For what concerns proof of stake, to the best of our knowledge, there are few papers that use formal methods to study the properties of the consensus protocols. Here, we first discuss contributions about PoS protocols in general and then we focus on the contributions about the Hybrid Casper protocol.

In [58], the authors conducted a study on Tendermint [81] which is a proof of stake consensus algorithm. They verify that the consensus protocol is deadlock-free and is able to reach consensus when at least 2/3 of the network is in agreement. They also prove that a minority set of nodes that compose more than 1/3 of the network is enough to censor the majority of the network and prevent the network from reaching consensus and conclude that the algorithm has some shortcomings on availability. The network of nodes is modelled as a parallel composition of processes and the system is verified by means the `PAT` model checker [80], which is neither stochastic nor probabilistic.

A first machine checked proof that guarantees both safety and liveness for a consensus algorithm is presented in [82]. They verify a Proof of Stake (PoS) Nakamoto-style blockchain protocol, using the foundational proof assistant Coq. In particular, they consider a PoS NSB in a synchronous network with a static set of corrupted parties. They define execution semantics for this setting and prove chain growth, chain quality, and common prefix. The chain growth property states that that the length of the best chain of an honest party increases over time. Chain quality says that within a sufficiently

large consecutive number of blocks of a main chain, some of them must be honest. The fact that there is a common prefix between the main chains of the honest nodes of the network is guaranteed by the common prefix property. These three properties together imply both safety (all honest parties reach the same decision) and liveness (a decision is reached eventually).

Coq has been also used to verify the Algorand PoS protocol [41]. Its correctness, *i.e.* two different blocks can never be certified in the same round even when the adversary completely control the network (asynchronous safety), is also demonstrated taking into account network delays, timing issues, and malicious nodes.

A PoS protocol with rigorous security guarantees is Ouroboros [49], where persistence and liveness properties are thoroughly studied. In that context, persistence means that, once an honest node declares a given transaction as "stable", then all the other honest nodes will agree on that choice; liveness means that once an honestly generated transaction has been made available to the network then it will become eventually stable. The authors prove that Ouroboros enjoys these properties in presence of adversaries through a manual proof. In our work we considered a probabilistic version of the safety and liveness property for Hybrid Casper and we proved them automatically through a model checker. We are confident that we can adopt our methodology to study similar properties of Ouroboros as well.

8.3.1 Hybrid Casper Related Works

The initial contribution of Hybrid Casper by Buterin et al. [21] also addresses the analysis of few properties. In particular, it is proved that the incentive mechanisms entail liveness and provide safety guarantees for the protocol. They also discuss issues related to parametrisation, funding, throughput and network overhead, and point out potential limitations of the protocol. The properties are analysed by means of numerical arguments on the states of possible computations. Our technique is different: we use statistical model checking to analyse the infinite-state blockchain model. By means of this model, we automatically verify quantitative properties, in particular those regarding the reachability of certain states of the network, and we can check them with different setting of protocol parameters. Moreover, it is possible to grasp other properties with

slight changes of the model, e.g. we need blocks structured as a sequence of transactions and a mechanism to count transactions therein.

Safety of Casper has been formally proved using the Isabelle proof assistant [63] and in the Coq theorem prover [67] by verifying basic properties about the ordering of messages, of justifications, and the state transitions. Similarly to these papers, our model overlooks the implementation details that do not affect the properties of interest (such as the process of creating new blocks). However, in the foregoing contributions, relevant properties, such as accountable safety and (a form of) liveness are demonstrated in the case when the set of validators is dynamic and at least 2/3 of them behave honestly [4]. In that case, different settings require completely new proofs. On the contrary, our analysis can be easily adapted to different settings of the protocol by changing basic parameters such as the network delay or adding/removing parallel processes acting as attackers. As said above, in our setting, safety and liveness have been proved in a probabilistic version.

Chapter 9

Conclusions

In this dissertation, we have discussed our efforts to enhance the PRISM model checker, resulting in the development of PRISM+, as well as our formal modelling and analysis of the Bitcoin Proof of Work protocol and the Hybrid Casper Ethereum protocol.

With the extension of PRISM we obtained a model checker able to analyse blockchain protocols. In particular, this has been possible thanks to the introduction of dynamic data types and the operations upon them.

For the Bitcoin protocol, we have studied and formalised with a theorem the probability that the ledger may devolve inconsistent copies because of forks. Two cases have been analysed: the first one is when the system consists of honest miners; the second one is when the system has a hostile node that mines blocks in the wrong positions. Our probabilities are parametric with respect to the number of nodes, their hashing power and the latency of the network.

Moreover, after defining the model in PRISM+, we performed some probabilistic analyses covering different features of the protocol. The first analysis was instrumental to assess the coherence of our model by verifying that the probability of mining a new block within a given amount of time and that of reaching a fork correspond to those of the real Bitcoin system and coincide with the values available in the literature. The second analysis was concerned with the trade-off between security and the difficulty of the cryptopuzzle. It has been observed that a slight decrease of the difficulty level of the cryptopuzzle leads to a significant increase in the speed of mining at the cost of an almost irrelevant increase

in the probability of a fork. We also modelled and analysed networks with churn nodes, which provide a more realistic account of the behaviour of this complex platform. In particular, we pointed out that in the scenarios that we investigated, churn nodes have a strong impact on the way the mining intervals vary with time: indeed, when a node leaves the network frequently, there is an immediate consequence on how the hashing power is distributed in the network. Finally, we simulated the Bitcoin protocol taking into account different kinds of network topologies. The driving question was checking whether the considered alternative topologies have resistance to forks equal to or greater than the original one of Bitcoin. The results of our simulations pointed out that the fewer the nodes are connected, the higher the probability of reaching a state of fork. Moreover, our result made evident that the dynamic participation of nodes affects the process of data propagation in the network. Namely, when a node disconnects, all of its connections are deactivated and network connectivity is reduced and this leads to an increase in the mean number of hops required for a block of transactions to propagate across the network.
For what concerns the Hybrid Casper protocol, we proved our model coherent with respect to the results presented in [20]. Also in this case, we have exploited the automatic verification machinery of `PRISM+` to perform several probabilistic analyses and to study the protocol behaviour in different settings of the basic parameters, such as rate of creation of blocks and penalty strategies. Our results have shown that increasing the rate of creation severely impacts on the justification/finalisation of blocks and have confirmed that higher is the penalty over the stake, lower is the rate of misbehaviour. We have also studied the behaviour of Hybrid Casper against two well known attacks: the eclipse and the majority attacks. Our results confirm that the protocol is robust against these two attacks when the original Hybrid Casper parameters are considered.

The abstraction of the adversary used in this dissertation to model the attack towards the Bitcoin blockchain is not the best one an adversary can implement, but the analysis of further strategies is left to future work. An interesting topic of future research could be the combination of the churn nodes model and different kinds of topologies to analyse how this could affect the likelihood of a fork. In future research, we also plan to study security issues by considering both peer-to-peer network-based attacks and mining-based

attacks. The first type of attacks, e.g. Eclipse attack [44] and Sybil attack [31], can be modelled by changing the behaviour of the network process, whereas the second one, e.g. 51% attack, can be analysed by introducing malicious miner processes. Our current PRISM+ model renders validators as pure stochastic processes and abstracts away the fact that they are actually rational agents that compete with each other to maximise a given utility function. Therefore, their behaviour is not only driven by the protocol but also by a given strategy that they follow when finalising blocks. In such a competitive setting, the strategy of a validator may also depend on the ones of others and validators may group themselves in coalitions that may collaborate to achieve a common goal. We plan to investigate these scenarios by extending the model with strategies and possible collaborative behaviours. Presumably, this will require the formalisation of utility functions [50] and will require particular care because PRISM (and PRISM+) manifests scalability issues when there are a lot of processes running in parallel (which is the case when coalitions are modelled). Since the maximum block size indirectly defines the maximum number of transactions carried within a block, large blocks may cause slower propagation speeds, which in turn increases the stale block rate. Thus, it could be interesting to extend our model to take into account the block size and verify how different sizes impact the security of the system. Moreover, we plan to apply our approach to the current Casper protocol. More specifically, we plan to study the *random validation mechanism*, which Casper uses to select the validator who can propose a new block [19]. This mechanism seems to be a perfect test-bed for our technique: the rapid analysis of the random protocol may help in taking the right decisions. For example, the simple technique where the hash of the previous block functions is used as a random seed for leader selection on the next round has been already proven to be vulnerable to attacks [1]. It is also on our agenda the comparison of Hybrid Casper and Casper to quantify which one provides a better trade-off between security and performance and to estimate their resilience to possible attacks. We finally plan the analysis of other recent Byzantine fault tolerant protocols such as Algorand [41] and Tendermint [58] and quantitatively compare them to (Hybrid) Casper.

Consensus protocols are important for ensuring the security and reliability of blockchain systems, but they are not the only way to achieve this goal. Since we studied

the smart contract implemented in Hybrid Casper, we realised that one of the key features of blockchain systems is their ability to store and execute smart contracts. These contracts, which are agreements between different parties, are used to codify the exchange of resources and are becoming increasingly important for storing security-critical assets. However, smart contract languages can be difficult to understand and smart contracts are vulnerable to unexpected attacks [6]. Therefore, it is crucial to ensure the comprehensibility and security of smart contracts in order to achieve trustworthiness and prevent vulnerabilities and attacks, such as the DAO attack. In this context applies our proposed solution to these challenges and our current theme of research consists of a language-independent modelling framework for smart contracts. The high abstraction level inherent in a modelling language fosters comprehensibility and, therefore, validation of the behaviour of smart contracts. The proposed modelling language features succinct abstractions of the concepts underlying many of the distributed ledger technologies based on blockchains. Moreover, a modelling language permits to abstract away from low-level concerns, which makes it possible to formally prove and certify the absence of several classes of attacks with a high degree of automation. The correctness of properties of smart contracts will be expressed relative to a formal operational semantics. Our approach will permit the verification of datacentric correctness properties (functional correctness), but also security properties of the modelled smart contracts. The main goal is to obtain an executable smart contract modelling language that is equipped with an operational formal semantics, sufficiently flexible and modular to represent security relevant phenomena, such as reentry or state reverting operations, and provides a consolidated and unified view on the fundamental language concepts that constitute blockchain-based distributed ledgers. We are working on a translator that converts existing legacy smart contracts into models (and vice versa). The plan is to obtain a deductive verification framework and static analysis tool to prove the absence of relevant classes of security vulnerabilities and functional correctness of the smart contracts.

List of Figures

Listings

Bibliography

[1] Mansoor Ahmed and Kari Kostiainen. "Don't Mine, Wait in Line: Fair and Efficient Blockchain Consensus with Robust Round Robin." In: *arXiv: Cryptography and Security* (2018).

[2] Amani Altarawneh et al. "Availability analysis of a permissioned blockchain with a lightweight consensus protocol." In: *Computers & Security* 102 (2021), p. 102098. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2020.102098. URL: https://www.sciencedirect.com/science/article/pii/S0167404820303710.

[3] Cambridge Center for Alternative Finance. *Cambridge Bitcoin Electricity Consumption Index*. https://cbeci.org/. (last access 2021). 2021.

[4] Musab A. Alturki et al. *Verifying Gasper with Dynamic Validator Sets in Coq*. Tech. rep. 2020.

[5] Andreas M Antonopoulos and Gavin Wood. *Mastering ethereum: building smart contracts and dapps*. O'reilly Media, 2018.

[6] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. "A Survey of Attacks on Ethereum Smart Contracts (SoK)." In: *Principles of Security and Trust - 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*. Ed. by Matteo Maffei and Mark Ryan. Vol. 10204. Lecture Notes

in Computer Science. Springer, 2017, pp. 164–186. DOI: 10.1007/978-3-662-54455-6_8. URL: https://doi.org/10.1007/978-3-662-54455-6_8.

[7] Adnan Aziz et al. "Verifying Continuous Time Markov Chains." In: vol. 1102. Lecture Notes in Computer Science. Springer. Computer Aided Verification, 8th International Conference, CAV '96., 1996, pp. 269–276. DOI: 10.1007/3-540-61474-5_75. URL: https://doi.org/10.1007/3-540-61474-5_75.

[8] C. Baier et al. "Model-checking algorithms for continuous-time Markov chains." In: *IEEE Transactions on Software Engineering* 29.6 (2003), pp. 524–541. DOI: 10.1109/TSE.2003.1205180.

[9] Christel Baier et al. "Model Checking Continuous-Time Markov Chains by Transient Analysis." In: *Computer Aided Verification*. Ed. by E. Allen Emerson and Aravinda Prasad Sistla. Springer Berlin Heidelberg. Berlin, Heidelberg, 2000, pp. 358–372. ISBN: 978-3-540-45047-4.

[10] Martijn Bastiaan. "Preventing the 51%-Attack: a Stochastic Analysis of Two Phase Proof of Work in Bitcoin." In: 2015.

[11] Johan Bengtsson et al. "UPPAAL—a Tool Suite for Automatic Verification of Real-Time Systems." In: Proceedings of the DIMACS/SYCON Workshop on Hybrid Systems III: Verification, Control: Verification, and Control. New Brunswick, NeW Jersey, USA: Springer-Verlag, 1996, 232–243. ISBN: 354061155X.

[12] Bruno Biais et al. "The blockchain folk theorem." In: *The Review of Financial Studies* 32.5 (2019), pp. 1662–1715.

[13] Stefano Bistarelli et al. *PRISM+ software package, supporting material, and additional experiments*. 2022. URL: https://github.com/adeleveschetti/bitcoin-analysis (visited on 12/06/2022).

[14] Stefano Bistarelli et al. "End-to-End Voting with Non-Permissioned and Permissioned Ledgers." In: *J. Grid Comput.* 17.1 (2019), pp. 97–118. DOI: 10.1007/s10723-019-09478-y. URL: https://doi.org/10.1007/s10723-019-09478-y.

[15] *Bitcoin*. https://bitcoin.org/en/.

[16] *Blackcoin*. https://blackcoin.org/.

[17] R. Bowden et al. "Block arrivals in the Bitcoin blockchain." In: *CoRR* abs/1801.07447 (2018).

[18] Vitalik Buterin. *Ethereum White Paper*. https://github.com/ethereum/wiki/wiki/White-Paper. 2013.

[19] Vitalik Buterin and V. Griffith. "Casper the Friendly Finality Gadget." In: *ArXiv* abs/1710.09437 (2017).

[20] Vitalik Buterin et al. "Combining GHOST and Casper." In: *CoRR* abs/2003.03052 (2020).

[21] Vitalik Buterin et al. "Incentives in Ethereum's hybrid Casper protocol." In: *International Journal of Network Management* 30.5 (2020), e2098.

[22] Kaylash Chaudhary et al. "Modeling and Verification of the Bitcoin Protocol." In: vol. 196. EPTCS. MARS. 2015, pp. 46–60.

[23] Huashan Chen et al. "A Survey on Ethereum Systems Security: Vulnerabilities, Attacks, and Defenses." In: *ACM Comput. Surv.* 53.3 (2020). ISSN: 0360-0300. DOI: 10.1145/3391195. URL: https://doi.org/10.1145/3391195.

[24] Herman Chernoff. "A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the sum of Observations." In: *Annals of Mathematical Statistics* 23 (1952), pp. 493–507.

[25] E. M. Clarke, E. A. Emerson, and A. P. Sistla. "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications." In: *ACM Trans. Program. Lang. Syst.* 8.2 (1986), 244–263. ISSN: 0164-0925. DOI: 10.1145/5397.5399. URL: https://doi.org/10.1145/5397.5399.

[26] Alexandre David et al. "Uppaal SMC Tutorial." In: *Int. J. Softw. Tools Technol. Transf.* 17.4 (Aug. 2015), 397–415. ISSN: 1433-2779. URL: https://doi.org/10.1007/s10009-014-0361-y.

[27] Christian Decker and Roger Wattenhofer. "Information propagation in the Bitcoin network." In: *13th IEEE International Conference on Peer-to-Peer Computing, IEEE P2P 2013, Trento, Italy, September 9-11, 2013, Proceedings*. IEEE, P2P 2013. 2013, pp. 1–10. DOI: 10.1109/P2P.2013.6688704. URL: https://doi.org/10.1109/P2P.2013.6688704.

[28] Evangelos Deirmentzoglou, Georgios Papakyriakopoulos, and Constantinos Patsakis. "A Survey on Long-Range Attacks for Proof of Stake Protocols." In: *IEEE Access* 7 (2019), pp. 28712–28725. DOI: 10.1109/ACCESS.2019.2901858.

[29] Giorgio Delzanno, Michele Tatarek, and Riccardo Traverso. "Model Checking Paxos in Spin." In: *Proceedings Fifth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2014, Verona, Italy, September 10-12, 2014*. Ed. by Adriano Peron and Carla Piazza. Vol. 161. EPTCS. 2014, pp. 131–146. DOI: 10.4204/EPTCS.161.13. URL: https://doi.org/10.4204/EPTCS.161.13.

[30] Max DiGiacomo-Castillo et al. "Model Checking Bitcoin and other Proof-of-Work Consensus Protocols." In: 2020 IEEE International Conference on Blockchain (Blockchain). 2020, pp. 351–358. DOI: 10.1109/Blockchain50366.2020.00051.

[31] John R. Douceur. "The Sybil Attack." In: vol. 2429. Lecture Notes in Computer Science. Springer, IPTPS 2002. Peer-to-Peer Systems, First International Workshop, IPTPS 2002., 2002, pp. 251–260. DOI: 10.1007/3-540-45748-8_24. URL: https://doi.org/10.1007/3-540-45748-8_24.

[32] *Ethereum*. https://www.ethereum.org/.

[33] Ittay Eyal and Emin Gün Sirer. "Majority is Not Enough: Bitcoin Mining is Vulnerable." In: *Commun. ACM* 61.7 (June 2018), pp. 95–102. ISSN: 0001-0782. DOI: 10.1145/3212998. URL: http://doi.acm.org/10.1145/3212998.

[34] Dean Fantazzini and Nikita Kolodin. "Does the Hashrate Affect the Bitcoin Price?" In: *Journal of Risk and Financial Management* 13.11 (2020). ISSN: 1911-8074. DOI: 10.3390/jrfm13110263. URL: https://www.mdpi.com/1911-8074/13/11/263.

[35] Carlos Faria and Miguel Correia. "BlockSim: Blockchain Simulator." In: *2019 IEEE International Conference on Blockchain (Blockchain)*. 2019, pp. 439–446.

[36] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. "Impossibility of Distributed Consensus with One Faulty Process." In: *J. ACM* 32.2 (1985), pp. 374–382. DOI: 10.1145/3149.214121. URL: https://doi.org/10.1145/3149.214121.

[37] Ulrich Gallersdörfer, Lena Klaaßen, and Christian Stoll. "Energy consumption of cryptocurrencies beyond bitcoin." In: *Joule* 4.9 (2020), pp. 1843–1846.

[38] Letterio Galletta et al. *PRISM+ software package, supporting material, and additional experiments*. 2022. URL: https://github.com/adeleveschetti/casper-analysis (visited on 03/05/2022).

[39] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. "The Bitcoin Backbone Protocol: Analysis and Applications." In: vol. 9057. Lecture Notes in Computer Science. Springer. Advances in Cryptology - EUROCRYPT 2015 - 34th Annual

International Conference on the Theory and Applications of Cryptographic Techniques., 2015, pp. 281–310. DOI: 10.1007/978-3-662-46803-6_10. URL: https://doi.org/10.1007/978-3-662-46803-6_10.

[40] Arthur Gervais et al. "On the Security and Performance of Proof of Work Blockchains." In: *IACR Cryptol. ePrint Arch.* (2016), p. 555. URL: http://eprint.iacr.org/2016/555.

[41] Yossi Gilad et al. "Algorand: Scaling Byzantine Agreements for Cryptocurrencies." In: *SOSP*. ACM, 2017, pp. 51–68.

[42] Vincent Gramoli. "From blockchain consensus back to Byzantine consensus." In: *Future Gener. Comput. Syst.* 107 (2020), pp. 760–769. DOI: 10.1016/j.future.2017.09.023. URL: https://doi.org/10.1016/j.future.2017.09.023.

[43] Stuart Haber and W. Scott Stornetta. "How to Time-Stamp a Digital Document." In: vol. 537. Lecture Notes in Computer Science. Springer. CRYPTO., 1990, pp. 437–455. DOI: 10.1007/3-540-38424-3_32. URL: https://doi.org/10.1007/3-540-38424-3_32.

[44] Ethan Heilman et al. "Eclipse Attacks on Bitcoin's Peer-to-Peer Network." In: *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*. Ed. by Jaeyeon Jung and Thorsten Holz. USENIX Association, 2015, pp. 129–144. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman.

[45] Ethan Heilman et al. "Eclipse Attacks on Bitcoin's Peer-to-Peer Network." In: *Proceedings of the 24th USENIX Conference on Security Symposium*. SEC'15. USA: USENIX Association, 2015, 129–144. ISBN: 9781931971232.

[46] Charles Antony Richard Hoare. "Communicating sequential processes." In: *Communications of the ACM* 21.8 (1978), pp. 666–677.

[47] Wassily Hoeffding. "Probability Inequalities for sums of Bounded Random Variables." In: *The Collected Works of Wassily Hoeffding*. Ed. by N. I. Fisher and P. K. Sen. New York, NY: Springer New York, 1994, pp. 409–426. ISBN: 978-1-4612-0865-5. DOI: 10.1007/978-1-4612-0865-5_26. URL: https://doi.org/10.1007/978-1-4612-0865-5_26.

[48] Wolfgang Jeltsch. "A Process Calculus for Formally Verifying Blockchain Consensus Protocols." In: *Declarative Programming and Knowledge Management - Conference on Declarative Programming, DECLARE 2019, Unifying INAP, WLP, and WFLP, Cottbus, Germany, September 9-12, 2019, Revised Selected Papers*. Ed. by Petra Hofstedt et al. Vol. 12057. Lecture Notes in Computer Science. Springer, 2019, pp. 24–39. DOI: 10.1007/978-3-030-46714-2_2. URL: https://doi.org/10.1007/978-3-030-46714-2_2.

[49] Aggelos Kiayias et al. "Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol." In: *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10401. Lecture Notes in Computer Science. Springer, 2017, pp. 357–388. DOI: 10.1007/978-3-319-63688-7_12. URL: https://doi.org/10.1007/978-3-319-63688-7_12.

[50] Marta Kwiatkowska, David Parker, and Clemens Wiltsche. "PRISM-Games: Verification and Strategy Synthesis for Stochastic Multi-Player Games with Multiple Objectives." In: 20.2 (Apr. 2018), 195–210. ISSN: 1433-2779. DOI: 10.1007/s10009-017-0476-z. URL: https://doi.org/10.1007/s10009-017-0476-z.

[51] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. "PRISM 4.0: Verification of Probabilistic Real-Time Systems." In: vol. 6806. Lecture Notes in Computer Science. Springer. Computer Aided Verification - 23rd International Conference,

CAV 2011., 2011, pp. 585–591. DOI: 10.1007/978-3-642-22110-1_47. URL: https://doi.org/10.1007/978-3-642-22110-1_47.

[52] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. "Probabilistic Symbolic Model Checking with PRISM: A Hybrid Approach." In: Lecture Notes in Computer Science 2280 (2002), pp. 52–66. DOI: 10.1007/3-540-46002-0_5. URL: https://doi.org/10.1007/3-540-46002-0_5.

[53] Marta Z. Kwiatkowska, Gethin Norman, and Roberto Segala. "Automated Verification of a Randomized Distributed Consensus Protocol Using Cadence SMV and PRISM." In: *Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings*. Ed. by Gérard Berry, Hubert Comon, and Alain Finkel. Vol. 2102. Lecture Notes in Computer Science. Springer, 2001, pp. 194–206. DOI: 10.1007/3-540-44585-4_17. URL: https://doi.org/10.1007/3-540-44585-4_17.

[54] Leslie Lamport. "Fast Paxos." In: *Distributed Comput.* 19.2 (2006), pp. 79–103. DOI: 10.1007/s00446-006-0005-x. URL: https://doi.org/10.1007/s00446-006-0005-x.

[55] Jingming Li et al. "Energy consumption of cryptocurrency mining: A study of electricity consumption in mining cryptocurrencies." In: *Energy* 168 (2019), pp. 160–168.

[56] Quan-Lin Li et al. "Markov Processes in Blockchain Systems." In: *CoRR* abs/1904.03598 (2019). arXiv: 1904.03598. URL: http://arxiv.org/abs/1904.03598.

[57] Zhiqiang Liu et al. "Fork-free hybrid consensus with flexible Proof-of-Activity." In: *Future Generation Computer Systems* 96 (2019), pp. 515–524. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2019.02.059.

URL: https://www.sciencedirect.com/science/article/pii/S0167739X18326256.

[58] Wai Yan Maung Maung Thin et al. "Formal Analysis of a Proof-of-Stake Blockchain." In: *2018 23rd International Conference on Engineering of Complex Computer Systems (ICECCS)*. 2018, pp. 197–200. DOI: 10.1109/ICECCS2018.2018.00031.

[59] Saeideh Gholamrezazadeh Motlagh, Jelena V. Misic, and Vojislav B. Misic. "An analytical model for churn process in Bitcoin network with ordinary and relay nodes." In: *Peer-to-Peer Networking and Applications* 13.6 (2020), pp. 1931–1942. DOI: 10.1007/s12083-020-00953-y. URL: https://doi.org/10.1007/s12083-020-00953-y.

[60] Saeideh Gholamrezazadeh Motlagh, Jelena V. Misic, and Vojislav B. Misic. "Impact of Node Churn in the Bitcoin Network." In: *IEEE Trans. Netw. Sci. Eng.* 7.3 (2020), pp. 2104–2113. DOI: 10.1109/TNSE.2020.2974739. URL: https://doi.org/10.1109/TNSE.2020.2974739.

[61] Saeideh Gholamrezazadeh Motlagh, Jelena V. Misic, and Vojislav B. Misic. "Modeling of Churn Process in Bitcoin Network." In: *International Conference on Computing, Networking and Communications, ICNC 2020, Big Island, HI, USA, February 17-20, 2020*. IEEE. 2020, pp. 686–691. DOI: 10.1109/ICNC47757.2020.9049704. URL: https://doi.org/10.1109/ICNC47757.2020.9049704.

[62] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. https://bitcoin.org/bitcoin.pdf. 2008.

[63] R. Nakamura, T. Jimba, and D. Harz. "Refinement and Verification of CBC Casper." In: *2019 Crypto Valley Conference on Blockchain Technology (CVCBT)*. 2019, pp. 26–38. DOI: 10.1109/CVCBT.2019.00008.

[64] E. Napoletano and John Schmidt. *Decentralized Finance Is Building A New Financial System*. https://www.forbes.com/advisor/investing/defi-decentralized-finance/. (last access 2021). 2021.

[65] Till Neudecker and Hannes Hartenstein. "Short Paper: An Empirical Analysis of Blockchain Forks in Bitcoin." In: *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*. Ed. by Ian Goldberg and Tyler Moore. Vol. 11598. Lecture Notes in Computer Science. Springer, 2019, pp. 84–92. DOI: 10.1007/978-3-030-32101-7_6. URL: https://doi.org/10.1007/978-3-030-32101-7_6.

[66] *Nxt*. https://www.jelurida.com/nxt.

[67] Karl Palmskog et al. "Verification of Casper in the Coq Proof Assistant." In: 2018.

[68] Rafael Pass, Lior Seeman, and Abhi Shelat. "Analysis of the Blockchain Protocol in Asynchronous Networks." In: vol. 10211. Lecture Notes in Computer Science. Springer. EUROCRYPT., 2017, pp. 643–673. DOI: 10.1007/978-3-319-56614-6_22. URL: https://doi.org/10.1007/978-3-319-56614-6_22.

[69] Remigijus Paulavicius, Saulius Grigaitis, and Ernestas Filatovas. "A Systematic Review and Empirical Analysis of Blockchain Simulators." In: *IEEE Access* 9 (2021), pp. 38010–38028. DOI: 10.1109/ACCESS.2021.3063324. URL: https://doi.org/10.1109/ACCESS.2021.3063324.

[70] *Peercoin*. https://www.peercoin.net/.

[71] Andrea Pinna et al. "A Petri Nets Model for Blockchain Analysis." In: *Comput. J.* 61.9 (2018), pp. 1374–1388. DOI: 10.1093/comjnl/bxy001. URL: https://doi.org/10.1093/comjnl/bxy001.

[72] Pierre-Yves Piriou and Jean-Francois Dumas. "Simulation of Stochastic Blockchain Models." In: *14th European Dependable Computing Conference, EDCC 2018, Iaşi, Romania, September 10-14, 2018*. IEEE Computer Society, EDCC 2018. 2018, pp. 150–157. DOI: 10.1109/EDCC.2018.00035. URL: https://doi.org/10.1109/EDCC.2018.00035.

[73] George Pîrlea and Ilya Sergey. "Mechanising blockchain consensus." In: ed. by June Andronick and Amy P. Felty. CPP 2018, Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, Los Angeles, CA, USA, January 8-9, 2018. 2018, pp. 78–90. DOI: 10.1145/3167086. URL: https://doi.org/10.1145/3167086.

[74] *Quorum*. https://consensys.net/quorum/.

[75] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. "Optimal Selfish Mining Strategies in Bitcoin." In: *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*. Ed. by Jens Grossklags and Bart Preneel. Vol. 9603. Lecture Notes in Computer Science. Springer, 2016, pp. 515–532. DOI: 10.1007/978-3-662-54970-4_30. URL: https://doi.org/10.1007/978-3-662-54970-4_30.

[76] Koushik Sen, Mahesh Viswanathan, and Gul Agha. "Statistical model checking of black-box probabilistic systems." In: *International Conference on Computer Aided Verification*. Springer. 2004, pp. 202–215.

[77] Md. Atik Shahriar et al. "Modelling Attacks in Blockchain Systems using Petri Nets." In: *19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2020, Guangzhou, China, December 29, 2020 - January 1, 2021*. Ed. by Guojun Wang et al. IEEE, 2020, pp. 1069–1078.

DOI: 10.1109/TrustCom50675.2020.00142. URL: https://doi.org/10.1109/TrustCom50675.2020.00142.

[78] Yahya Shahsavari et al. "A Theoretical Model for Analysis of Firewalls Under Bursty Traffic Flows." In: *IEEE Access* 7 (2019), pp. 183311–183321. DOI: 10.1109/ACCESS.2019.2926925. URL: https://doi.org/10.1109/ACCESS.2019.2926925.

[79] Yonatan Sompolinsky and Aviv Zohar. "Bitcoin's Security Model Revisited." In: *CoRR* abs/1605.09193 (2016). arXiv: 1605.09193. URL: http://arxiv.org/abs/1605.09193.

[80] Jun Sun et al. "PAT: Towards Flexible Verification under Fairness." In: *Computer Aided Verification*. Ed. by Ahmed Bouajjani and Oded Maler. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 709–714.

[81] *Tendermint*. https://tendermint.com/.

[82] Søren Eller Thomsen and Bas Spitters. "Formalizing Nakamoto-Style Proof of Stake." In: *CoRR* abs/2007.12105 (2020). arXiv: 2007.12105. URL: https://arxiv.org/abs/2007.12105.

[83] Tatsuhiro Tsuchiya and Andre Schiper. "Model Checking of Consensus Algorit." In: *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*. 2007, pp. 137–148. DOI: 10.1109/SRDS.2007.20.

[84] Hrakan LS Younes and Reid G Simmons. "Probabilistic verification of discrete event systems using acceptance sampling." In: *International Conference on Computer Aided Verification*. Springer. 2002, pp. 223–235.

[85] Alexei Zamyatin et al. "Flux: Revisiting Near Blocks for Proof-of-Work Blockchains." In: *IACR Cryptology ePrint Archive* 2018 (2018), p. 415.

[86] Zixuan Zhang, Michael Zargham, and Victor M. Preciado. "On modeling blockchain-enabled economic networks as stochastic dynamical systems." In: *Appl. Netw. Sci.* 5.1 (2020), p. 19. DOI: 10.1007/s41109-020-0254-9. URL: https://doi.org/10.1007/s41109-020-0254-9.