

Resilience of Hybrid Casper under varying values of parameters

LETTERIO GALLETTA, IMT School for Advanced Studies Lucca, Italy

COSIMO LANEVE, University of Bologna, Italy

IVAN MERCANTI, IMT School for Advanced Studies Lucca, Italy

ADELE VESCHETTI, University of Bologna, Italy

Hybrid Casper is the new Ethereum blockchain protocol that uses both Proof of Work and Proof of Stake to reach a consensus between nodes. Here, we analyse the protocol using PRISM+, an extension of the probabilistic model checker PRISM with primitives for expressing blockchain data types. First, we extend PRISM+ to include data types and operations for modelling and analysing Proof of Stake-based consensus protocols. Then, we model Hybrid Casper in PRISM+ as a parallel composition of stochastic processes, thus precisely describing the behaviour of the protocol and highlighting its corner cases. PRISM+ is therefore used to rapidly and automatically analyse the resilience of Hybrid Casper when tuning, up or down, several basic parameters of the protocol, such as the rates of creating blocks, and the strategies for determining penalties. Finally, we study the robustness of Hybrid Casper to two well known attacks: the Eclipse attack and the majority attack.

CCS Concepts: • **Theory of computation** → **Verification by model checking**; • **Security and privacy** → **Logic and verification**.

Additional Key Words and Phrases: stochastic modelling and analysis, proof of stake, blockchain fork

1 INTRODUCTION

Blockchain is revolutionising 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 [38]), running decentralized applications (Ethereum smart contracts [11]), the implementation of voting systems [10] and Decentralized Finance [40].

Blockchain 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-based system must address is the consistency of the ledger upon updates performed by different nodes. To overcome this problem, these systems rely on a consensus protocol that imposes a total order on the updates performed by the nodes. Traditionally, following the seminal work by Nakamoto [38], these protocols have been based on a probabilistic mechanism called Proof of Work (PoW) whereby nodes can update the ledger only if they solve a hard computational problem.

Because of the hardness of the computational problem, PoW has the substantial short-come of requiring a very large amount of computational resources and energy [21]. 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

Authors' addresses: Letterio Galletta, letterio.galletta@imtlucca.it, IMT School for Advanced Studies Lucca, Lucca, Italy; Cosimo Laneve, cosimo.laneve@unibo.it, University of Bologna, Bologna, Italy; Ivan Mercanti, ivan.mercanti@imtlucca.it, IMT School for Advanced Studies Lucca, Lucca, Italy; Adele Veschetti, adele.veschetti2@unibo.it, University of Bologna, Bologna, Italy.

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

© 2022 Association for Computing Machinery.

2769-6472/2022/11-ART \$15.00

<https://doi.org/10.1145/3571587>

to the quantity of cryptocurrency they invested to be part of the network – the *stake*. One of these protocols – the Casper Protocol [12] – will be adopted by future releases of Ethereum. Meantime, to ensure a smooth transition with minimal impact on the users, Ethereum developers have deployed a hybrid version of Casper – the *Hybrid Casper Protocol* – that uses both PoW and PoS [14]. In particular, Hybrid Casper speeds up block creation by means of a less expensive PoW than Bitcoin (block creation occurs every 14 seconds in Hybrid Casper [14], while it takes 600 seconds in Bitcoin [17]) 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 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 *finalized*, which means that it is justified *and* its child checkpoint is justified as well. Finalization guarantees consistency of the corresponding blockchains in the distributed ledger.

Since Hybrid Casper is a recent proposal, the protocol may have corner cases that can pave the way to possible attacks. Therefore, in this paper, we present an analysis of Hybrid Casper by means of a formal model and an automatic verification technique that allow us (i) to predict how it behaves in different settings of the parameters; (ii) to understand its resilience and robustness to attacks; and (iii) to study new variants of the protocol. The approach we follow is the one of Bistarelli et al. [9], where the Bitcoin protocol has been analysed using PRISM+, an extension of the PRISM model checker [32] with primitives for PoW protocols, such as data types ledger, block and set, and the operations upon them. In particular, to cover Hybrid Casper, we have further extended PRISM+ with data types and operations that are typical of PoS protocols, such as the map data type to express the table of the stakes and the management of votes. Once the extension has been prototyped, the protocol is rendered as a *parallel composition* of PRISM+ processes where the time to create a block and to broadcast a message is an *exponential distribution* with a *rate parameter* associated to process actions. Henceforth, by tuning up and down the rates, it has been possible to *rapidly and automatically analyse* different settings of the basic parameters of the protocol and check the corresponding correctness, as done in, e.g., [6, 30]. In particular, we measure the impact of tuning the rates for creating blocks as well as the penalties that nodes have to pay when they vote maliciously and the resistance of the protocol to two well known attacks.

The main contributions of this paper are:

- We first extend PRISM+ with PoS data types and operations; the software package is available online [22] and can be used to model and study quantitative properties of generic PoW and PoS protocols.
- We then model Hybrid Casper as a PRISM+ process and verify that the model is compliant with the results of Buterin et al. [14] when the rate of actions are the same. In particular, we show that (i) the probability of justifying a checkpoint within one epoch is 0.672; and (ii) that the probability of finalizing a checkpoint within 20 epochs is almost 1.
- We give a stochastic characterization of the safety and liveness properties proposed in [14] 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/finalization of blocks. In particular, we show that when the creation rate is 6 seconds (instead of the standard value 14 seconds), the probability of justifying a block within 1 epoch is 0.005.
- We analyse different penalty strategies, by studying different quotas of penalty in case of misbehaviours of validators. We show that (i) when the penalty is 40% of the stake, the stake of a misbehaving validator is almost 0 ether after 17 epochs, even if it misvoted just 3 times; and (ii) when the penalty is 20% of the stake, validator's stake decreases less rapidly (if it misvoted 3 times its stake is 7 ether).
- 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

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

the network, and force them to work on an untruthful view of the blockchain. Our results show that the probability of a successful attack is 0.049. 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. In this case, we show that the probability of a successful attack is less than 10^{-4} .

The paper is organized as follows. Section 2 compares our technique with respect to other analyses. Section 3 contains an overview of the Hybrid Casper protocol and of its consensus algorithm; Section 4 provides a quick introduction to PRISM; Section 5 presents PRISM+, the extension of PRISM with the data types for blockchain system and the new operations we introduce to model PoS consensus algorithms. Our model of Hybrid Casper is defined in Section 6 and the analyses assessing its coherence are in Section 7. Section 8 presents our study of the resilience of Hybrid Casper to the changes of different parameters, such as the time needed to create a new block and different strategies of penalties. It also presents our results about the robustness of the protocol to two well known attacks. Section 9 compares our proposal with the literature and Section 10 draws some conclusions and discusses possible future work. Table 1 summarizes the notation and the symbols used in the paper.

2 REMARKS ABOUT OUR TECHNIQUE

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. 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 Section 9. There are three mainstream approaches for *automatic* analysing consensus protocols: testing, simulation and formal verification.

In the first 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 [14] tests the behaviour of Ethereum protocols in scenarios that are similar to the final one. It is frequent that testnets 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 testnet is often tedious, time-consuming, and costly. For these reasons, testers hardly reproduce a precise deployment environment due to limited financial and timing resources.

The second approach uses simulators [43], which implement the protocols by ad-hoc modules that try to reconstruct the overall behaviour on a single machine [20]. 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. [24] 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.

The third approach 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 [19, 34, 36, 46]. 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. Therefore, while being correct, some 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 validators 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. For example, PRISM runs the model generating a certain number of *samples* of execution paths and evaluates the property being checked along these paths to perform a statistical analysis. To bound the length of execution paths, the tool imposes a maximum length on the executions.

Overall, we think that automatic analyses have pros and cons. However, it is possible to use them all together, with the opportunity to spot a large number of bugs at the early stages of software development. In this view, we think that our technique adds a new axis to the analysis of blockchain protocols that may complement the other techniques.

3 THE HYBRID CASPER PROTOCOL

Ethereum [11] is a peer-to-peer asynchronous network whose state is maintained through a distributed ledger. This ledger is a tree of blocks with a pointer, called *handle*, to a *leaf block at maximal depth*; the *blockchain* is the sequence of blocks from the handle to the root, called *genesis block*; each block in the ledger has a height which is the length of the path from the block to the genesis block.

Hybrid Casper [14] is a new 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 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, 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).

²See the GitHub implementation at <https://github.com/ethereum/eth2.0-specs>

Justifications and finalizations. The goal of the voting process is to *justify* and *finalize* 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 finalized. (The root block of the ledger, the genesis block, is both justified and finalized by definition.) This mechanism ensures that a finalized 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 finalized, 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 penalized. This is achieved by adjusting validators' stakes according their own voting behaviour: when a checkpoint is finalized the stakes of validators who voted for it is increased by a positive interest rate r (see Section 8.2 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 finalized 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 it is can be punished by slashing its deposits. 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 the deposits of validators are updated only after checkpoints get finalized.

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 [15]. In the nothing-at-stake, the attacker generates multiple conflicting blocks to maximize its benefit without risking its stake. This kind of attack is not possible 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 generates a new branch starting from an earlier block to overtake the main chain. The core idea is that when stake-holders have sold their stake in the system, nothing prevents them from performing a history-rewrite attack [16]. In Hybrid Casper the blockchain up to the most recently finalized checkpoint will never be reverted guaranteeing that a posterior corruption attack can not be successful. In simple terms, a revision fork that finalizes blocks older than the last finalized block will be ignored, because all clients will have already seen a finalized 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 [12].

4 A QUICK INTRODUCTION TO PRISM

PRISM [32] 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 Kwiatkowska et al. [33] 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

³An 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.

```

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 1. A PRISM specification modelling a N-place queue and a server.

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 *synchronize* with the corresponding commands, i.e. the modules execute the commands with action *a* at the same time.

The example in Listing 1, taken from the documentation,⁴ 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. The modules *queue* (line 6 to 12) and *server* (line 14 to 19) synchronize 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 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 synchronize with *queue* through the action *serve*. After this synchronization the server updates its state to busy ($s'=1$). The rate of the synchronization is the product of the two individual rates (in this case, $\lambda \cdot \mu$). The second command (line 18) of *server* states that a busy server ($s=1$) may complete its task with rate *gamma*.

⁴<http://www.prismmodelchecker.org/manual/ThePRISMLanguage/Example2>

In this paper we focus on Continuous Time Markov Chains (CTMC) models that are transition systems (as the one above) where each transition is labelled by a positive real number, called *rate*. In particular, if the rate of a transition from a state s to s' is r then the probability of moving from s to s' within $t \geq 0$ time units is $1 - e^{-r \cdot t}$, that is rates are used as parameters of an exponential distribution. Note that, higher is the rate, higher is the probability to leave s in a given time. For example, in Hybrid Casper, the creation of blocks can be approximated by an exponential distribution with rate $1/14$ (because a block is created every 14 seconds) [14] and an exponential distribution also approximates the probability of delivering blocks across the network [17].

When a CTMC state has several exiting transitions, e.g. the above state $x=1$, then the probability of choosing one of them depends on their rates – this is known as *race condition*. For example, if r_1, \dots, r_n are the rates of transitions exiting from a state s (and entering on pairwise different states), then the probability of taking a transition with rate r_i is r_i/R , where $R = r_1 + \dots + r_n$. This is due to the fact that the minimum of two exponentially distributed random variables is still an exponentially distributed random variable with a rate equal to the sum of their rates.

In the following sections 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 Continuous Stochastic Logic (CSL) [3–5], which is an extension of temporal logic with a probabilistic operator. The formulas we use in this paper have always the form

$$P=?[F \leq t \text{ property}]$$

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

$$P=?[F \leq 10 \text{ "full"}].$$

To actually compute this probability, PRISM performs *model checking*. However, since models may bear infinite sets of executions (in general and in our case, in particular), PRISM does not undertake an exhaustive exploration of

the state-space and sticks to a so called *statistical model checking*, which combines model checking and statistical methods.

In a nutshell, given a model of the stochastic system and a formula ϕ representing the property to verify, PRISM generates a finite number of executions and evaluates them to determine the fraction of executions satisfying ϕ . This process computes an approximation q' of the actual probability q for the formula ϕ , and offers a probabilistic guarantee on the accuracy of q' . In particular, the resulting q' is such that the probability that the error of the approximation is too high is bounded by a constant chosen by the user and called *confidence level*.

In summary, the core idea of statistical model checking is to conduct partial analyses of the system, monitor them, and then decide whether the system satisfies the property or not with some degree of confidence.

5 THE EXTENSION PRISM+

PRISM+ extends PRISM by adding a native support for expressing and manipulating dynamic 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. In this paper, we also provide ad-hoc primitives for the PoS protocols such as votes, penalties, and rewards. This section describes the data types and the operations PRISM+ provides, and the implementation of PRISM+ as a fork of PRISM.

5.1 Data types and Operations

The data types that have been implemented in PRISM+ are block, ledger, set, and map. These data types, in particular ledger and set make models of PRISM+ be infinite state. Therefore we analyse them by means of the statistical model checking engine. Below, we present and discuss the representation of these types and provide a precise semantics of the corresponding operations.

Blocks, noted b, p, \dots , are triples $(v^n; p; h)$, where v is the *name* of the validator v who created the block; n is a unique numeric label; p , called *father*, is the name of another block which $(v^n; p; h)$ points to; h is the *height* of the block in the ledger. For instance, $(v_3^0; v_4^7; 3)$ is a block named v_3^0 , which is the first block created by v_3 , whose father is the block named v_4^7 and which is at height 3. The operations on blocks are:

- $\text{createB}(v, n, L)$ returns a block $(v^n; p; h)$, where p is the handle of the ledger L (see below) and h is the height of p plus one;
- $\text{isCP}(b)$ returns true if the block b is a checkpoint (*i.e.* its height is a multiple of 64), false otherwise;
- $\text{height}(b)$ returns the height of b .

Ledgers, noted L, L', \dots , are tuples $\langle T; f; p; K \rangle$ where T is a tree of blocks; f is the name of a block in T (the *last finalized block* of L); p is the name of a leaf block at maximal height in the subtree rooted at f , called the *handle* of L ; and K is a mapping from blocks in T to integers such that, if b has been inserted more recently than b' in T , then $K(b) > K(b')$. The root of T is called *genesis block* and is denoted by $(\text{genesis}^0; \text{genesis}^0; \emptyset)$. The *blockchain* of L is the sequence of blocks that starts from the handle and reaches the genesis.

The operations on ledgers are:

- $\text{canBeIns}(L, b)$ returns true if b can be inserted in L (*i.e.* if the father of b is in L), false otherwise;
- $\text{addBtoL}(L, b)$ inserts b in L and returns the updated ledger (precondition: $\text{canBeIns}(L, b) = \text{true}$); it also updates the mapping K and, in case, it updates the handle;
- $\text{lastCP}(L)$ and $\text{lastboCP}(L)$ return the last checkpoint and the last-but-one checkpoint in L , respectively;
- $\text{lastF}(L)$ returns the last finalized block in L , *e.g.* $\text{lastF}(\langle T, f, p, K \rangle) = f$.

- `updateHF(L, b)` takes a ledger $L = \langle T; f; p; K \rangle$ and a block b that has been finalized such that $K(b) > K(f)$ and returns $L' = \langle T; b; p'; K \rangle$ where p' is the leaf block in the subtree of b such that $K(b)$ is the greatest value.

A important notion for ledgers is that of fork defined as follows:

Definition 5.1. Let $L_1 = \langle T_1; f_1; p_1; K_1 \rangle, \dots, L_n = \langle T_n; f_n; p_n; K_n \rangle$ be a set of ledgers and let m be the maximal height of the handles p_1, \dots, p_n . Let also L_{i_1}, \dots, L_{i_k} be the ledgers in the above set with the handle at height m . We say that the set L_1, \dots, L_n has a fork of length $m - h$, where h is the length of the maximal common suffix of the blockchains of L_{i_1}, \dots, L_{i_k} .

The following operations are used to compute and verify forks:

- `calculateFork(L1, ..., Ln)` returns the length of the fork in L_1, \dots, L_n , 0 if there is no fork.
- `verifyCP(L1, ..., Ln)` returns true if $\text{lastCP}(L_1) = \dots = \text{lastCP}(L_n)$, false otherwise.

Sets, noted S, S', \dots , are collections of blocks, whose operations are almost standard:

- `get(S)` returns a block randomly extracted from the set S , the block is not removed from S ;
- `add(S, b)` returns $S \cup \{b\}$;
- `receive(S)` returns a block b extracted from S ;
- `remove(S, b)` returns $S \setminus b$;
- `isEmpty(S)` returns true if the set S is empty, false otherwise.

In our Hybrid Casper model we use them to store the blocks that have been received but have to be inserted in the local ledger.

Maps, noted H, H', \dots , are used to record the stakes of validators and the votes of checkpoints. We use the following operations:

- `addVote(H, b, v)` returns the update of H where the vote of v for b has been recorded;
- `updateS(H, H', b)` returns H updated with the rewards and penalties for each validator according to if they voted correctly for b or not; the votes are taken from H' . The techniques for rewarding and penalizing the validators are described by Buterin et al. [14];
- `isJust(b, H, H')` returns true if the sum of the stakes of the validators in H that have voted for b is greater than $2/3$ of the total stake of the system, false otherwise. Stakes are retrieved from H' .

5.2 The implementation of PRISM+

PRISM+ has required a significant programming effort in order to make the data types of Section 5.1 native. The main challenge has been the fact that PRISM is not designed to be extended with plug-ins. Thus, we have implemented our data types and operations modifying the PRISM source code and providing a new software package that includes our extensions.

To support the augmented syntax, we have first extended the PRISM parser to allow users to use the new operations as they were built-in. In particular the extension of PRISM expressions with the new types has led to extending the PRISM abstract class `Expression`. For example, the implementation of the data type `map` is done by the Java class `ExpressionMap` (see the snippet of code in Listing 2) whose fields are:

- `name` which represents the name of the map;
- `length` which is the length of the list of blocks;
- `votedBlocks` which is a list of pairs storing the votes of each block.

The constructor of `ExpressionMap` takes as parameters the name of the map and a list of pairs (validator, vote); it initializes `votedBlocks` with the votes – see Listing 2. Note that each vote is stored in the array position corresponding to the name of the validator, e.g. the vote of validator v_1 is stored in position 1 of the array.

```

1 public class ExpressionMap extends Expression{
2
3     private String name;
4     private int length;
5     private List<Pair> votedBlocks;
6
7     public ExpressionMap(String n, List<Pair> b){
8         name = n;
9         this.votedBlocks = new ArrayList<Pair>();
10        if(b!=null) {
11            for(int i=0; i<b.size(); i++) {
12                votedBlocks.add(b.get(i));
13            }
14        }
15        length = votedBlocks.size();
16    }
17 }

```

Listing 2. The ExpressionMap class.

The addition of the new operations to PRISM has required the extension of the class ExpressionFunc where the built-in operations are defined. For example, the operation `addVote(H, b, v)` is reported in Listing 3. (All the other operations are implemented following the same schema.) The operation first checks if the map storing the

```

1 private ExpressionMap addVote(ExpressionMap table, ExpressionBlock block, String validator){
2
3     int j = 0;
4     boolean found = false;
5     for(int i = 0; i<validator.length() and !found; i++) {
6         if(Character.isDigit(nameTmp.charAt(i))) {
7             j = i;
8             found = true;
9         }
10    }
11    String nameTmp2 = validator.substring(j);
12    int whichMiner = 0;
13    if(found) {whichMiner = Integer.parseInt(nameTmp2);}
14    boolean flag = false;
15    if(table.getVotedBlocks()!=null) {
16        for(int i = 0; i<table.getVotedBlocks().size(); i++) {
17            if(table.getVotedBlocks().get(i).getBlock().equals(block)){
18                flag = true;
19                table.addVote(i, whichMiner, stake);
20            }
21        }
22        if(!flag) {
23            table.addBlock(block, whichMiner, stake);
24        }
25    }
26    else {
27        table = new ExpressionMap();
28        table.addBlock(block, whichMiner, stake);
29    }
30    return table;
31 }

```

Listing 3. The implementation of addVote operation.

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 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.

6 THE DEFINITION OF HYBRID CASPER IN PRISM+

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 1.

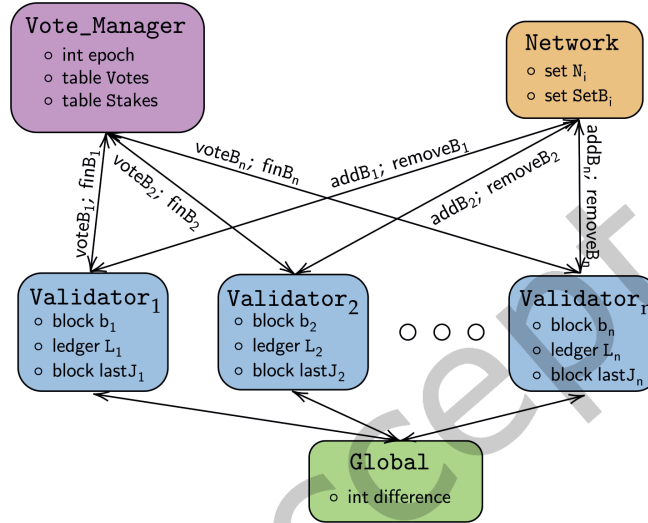


Fig. 1. The Ethereum PoS model architecture.

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 an auxiliary module that computes the length of forks – see Section 7. We note that the management of votes is *centralized* in Vote_Manager, which corresponds to the smart contract of Hybrid Casper [14].

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

- (1) *the network consists of validators that also create new blocks;*
- (2) *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;
- (3) *blocks are black boxes:* we omit any informations that is not relevant for the consensus, such as Solidity transactions.

For clarity sake, we present a sugared version of the PRISM+ code; the online repository [22] 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 Global module will be presented in the next section.

```

1 module Validator_i
2   STATE_i : [Start, Create, Receive, Move, Vote, Check, Fin] init Start;
3   L_i : ledger init ((genesis0; genesis0; 0); genesis0; genesis0; [genesis0 ↦ 1]);
4   c_i : [0..1000] init 0;
5   b_i : block;
6   lastJ_i : block init (genesis0; genesis0; 0);
7
8   [] (STATE_i=Start) ->
9     mR_i : (b_i'=createB(Validator_i, c_i, L_i)) & (c_i'=c_i+1) & (STATE_i'=Create);
10  [] (STATE_i=Start) -> hR_i : (STATE_i'=Receive);
11  [] (STATE_i=Start) -> rC_i : (b_i'=lastCP(L_i)) & (STATE_i'=Check);
12  [addB_i] (STATE_i=Create) & !isCP(b_i) ->
13    1 : (L_i'=addBtoL(L_i, b_i)) & (STATE_i'=Start);
14  [addB_i] (STATE_i=Create) & isCP(b_i) ->
15    1 : (L_i'=addBtoL(L_i, b_i)) & (STATE_i'=Vote);
16  [voteB_i] (STATE_i=Vote) -> 1 : (STATE_i'=Start);
17  [] (STATE_i=Receive) & !isEmpty(Set_i) ->
18    1 : (b_i'=receive(Set_i)) & (STATE_i'=Move);
19  [] (STATE_i=Receive) & isEmpty(Set_i) -> 1 : (STATE_i'=Start);
20  [removeB_i] (STATE_i=Move) & canBeIns(L_i, b_i) & isCP(b_i) ->
21    1 : (L_i'=addBtoL(L_i, b_i)) & (STATE_i'=Vote);
22  [removeB_i] (STATE_i=Move) & canBeIns(L_i, b_i) & !isCP(b_i) ->
23    1 : (L_i'=addBtoL(L_i, b_i)) & (STATE_i'=Start);
24  [] (STATE_i=Move) & !canBeIns(L_i, b_i) -> 1 : (STATE_i'=Start);
25  [] (STATE_i=Check) & isJust(b_i, Votes, Stakes) & lastJ_i=lastboCP(b_i, L_i) ->
26    1 : (lastJ_i'=b_i) & (L_i'=updateHF(L_i, lastJ_i)) & (STATE_i'=Fin);
27  [] (STATE_i=Check) & isJust(b_i, Votes, Stakes) & lastJ_i!=lastboCP(L_i) ->
28    1 : (lastJ_i'=b_i) & (STATE_i'=Start);
29  [] (STATE_i=Check) & !isJust(b_i, Votes) -> 1 : STATE_i'=Start;
30  [finB_i] (STATE_i=Fin) -> 1 : STATE_i'=Start;
31 endmodule

```

Listing 4. Pseudocode of a Validator.

6.1 Validator module

The code of Validator module is reported in Listing 4. A Validator is defined as a state machine with seven states. The current state is recorded in the variable `STATE_i` defined at line 2. The actions of Validator are guarded by `STATE_i` and update this variable when executed.

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 (see Section 5), they are associated with the rates mR_i , hR_i , and rC_i , respectively. The rates mR_i are defined as $1/s$, where s is the number of seconds needed to create a new block, since $s = 14$ in Hybrid Casper [14], then $mR_i = 1/14$. The rates hR_i are complementary to mR_i , therefore they are defined as $hR_i = 1 - mR_i$ (the rationale behind this choice is that a validator is more likely not to create a block rather than to create one). The rates rC_i represent how often a validator should check for new justified/finalized blocks. According to Buterin et al. [14], this happens at the end of each epoch, thus $rC_i = 1/(len_{epoch} \times s)$.

When Validator creates a new block, it updates its ledger and sends the created block to 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 6 of Listing 5). In our model, we set r_b to either $1/12.6$, which is the broadcast delay of the Bitcoin network [17] 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 [13]. 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 synchronizing with Vote_Manager through the action `voteB_i`; this synchronization 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 in case it 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 (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 r_{C_i} , 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 finalized and C_L becomes justified – in lines 25-26 this is performed by storing C_L in `lastF_i` and updating the last finalized block of L_i to `lastJ_i`. At the same time, the handle of L_i may be updated as well – see definition of `updateHF(L_i, lastJ_i)` in Section 6. 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.

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. On the opposite, an orphan block is modelled as a block that has not 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 `setMiner_i` of Network (see the discussion below).

6.2 Network module

The module Network is defined in Listing 5 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. In this section, we assume that validators are totally connected, therefore `N_i` is always equal to $\{1, \dots, i-1, i+1, \dots, n\}$. Network module synchronises with the validator i who creates a block by synchronizing on the action `addB_i`. When this happens, Network updates the sets of blocks of the validators contained in `N_i` (line 6). When a block has been added to the local ledger of `Validator_i`, by synchronizing with the action `removeB_i`, Network removes the block from the corresponding set (line 8).

```

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

```

Listing 5. Pseudocode of the Network.

6.3 Vote Manager module

The module Vote_Manager is reported in Listing 6. Its internal state consists of a map `Stakes` that stores the stakes of validators; a map `Votes` that takes the name of a block and return the list of validators who have voted for the corresponding block; and the epoch that records the height of the last finalized block. The module synchronizes with the validator i on actions `voteB_i` and `finB_i`. The synchronization on `voteB_i` adds the vote for the block that is stored in `b_i` to `Votes`, i.e. the name `Validator_i` is added to the list of `b_i` (line 8).

The synchronization on `finB_i` is used to compute the rewards and penalties for each validator when a block is finalized. In particular, this happens when the first validator finalizes a block `b` because, in this case, the height of `b` is higher than `epoch`. (This is our modelling of Hybrid Casper’s epochs.) In this case, both `Stakes` and `epoch` are updated with the new stakes and `height(b)`, respectively (lines 9–10). If the validator is not the first to finalize then no update occurs.

```

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

```

Listing 6. Pseudocode of the `Vote_Manager`.

7 COHERENCE OF THE HYBRID CASPER MODEL

The goal of this section is to validate our model with respect to literature [13, 14] on Hybrid Casper. To this aim, we set the basic parameters 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_i 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_i to the corresponding values (we haven’t undertaken such analyses because we miss the values).

The experiments that we describe in the rest of the paper were carried out on a Virtual Machine with 8 VCPU and 64 GB RAM. We set the PRISM+ model checker to generate 100000 samples of protocol executions. The verified systems were composed by n validators, with $n = 6, 8, 10, 12, 14, 16$, and the experiments are always run until we observe a stabilization of 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.

We start by computing the probabilities for creating a new block. Figure 2a reports these probabilities, which are calculated by letting PRISM+ to analyze the property:

$$P = ?[F \leq T \text{ "someCreated"}]$$

where `someCreated` is a label that identifies all the states in which a validator is in the state `Create`. In this figure and in Figure 2b, the broadcast delay r_b is $1/7$, that is, we assume that the time to deliver messages is $1/2$ of the time to create a block, *c.f.* (1/2)-network synchrony [13]. The results we obtain are coherent with the ones in the literature [13]; 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 following module `Global`

```

module Global
  difference : [0..N] init 0;
  [] (STATE_1 = Start | ... | STATE_n = Start) -> 1 :
    (difference' = calculateFork(L_1, ..., L_n))
endmodule

```

that computes the difference between the ledgers of the system every time one of them is modified (when the Validator_i changes its state to Start). To this aim, Global invokes the operation calculateFork defined in Section 5 that stores the length of the fork in the state variable difference. Therefore, following Section 4, the probability of reaching a state with a fork of length k within the first T time units is defined by the following formula:

$$P = ?[F \leq T \text{ difference} = k].$$

Figure 2b 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 $k \cdot 14$ 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 2c and 2d respectively report the probabilities of justification and finalization within k epochs. These

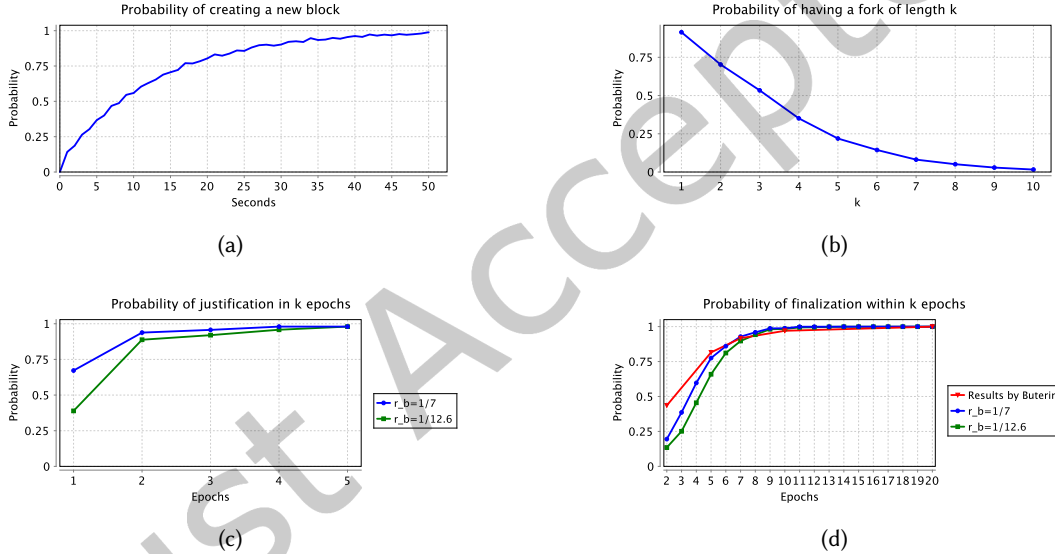


Fig. 2. Results of the analyses for assessing the coherence of the model.

probabilities are computed by PRISM+ using the formulas

$$P = ?[F \leq T \text{ "someJustified"}] \quad P = ?[F \leq T \text{ "someFinalized"}]$$

where someJustified and someFinalized are the labels that identify all the states in which a validator is in the state Check and the block is justified and finalized, respectively (lines 25-28 of Listings 4). The experiments reported in Figures 2c and 2d have been run with two different broadcast delays: the standard broadcast delay of Bitcoin, *i.e.* $r_b = 1/12.6$ [17] 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. Hence the voting process is longer and the probability is smaller when $r_b = 1/12.6$. 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 [13] 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 2d we report the probability of finalization within k epochs that have been computed by Buterin et al. [13] (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 [13] 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, Buterin et al. [13] proved properties of safety and liveness for Hybrid Casper. In particular, they define a protocol to be

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

Figures 3a and 3b show the analyses of safety and liveness in PRISM+ with respect to the broadcast delay r_b . In these analyses we adopt the setting of Buterin et al. [13], 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 finalizing two conflicting checkpoints is always 0 (Figure 3a) and the probability of finalizing a new checkpoint is always greater than 0.85 when $r_b \geq 1/100$ (it is almost 1 with $r_b \geq 1/12.6$).

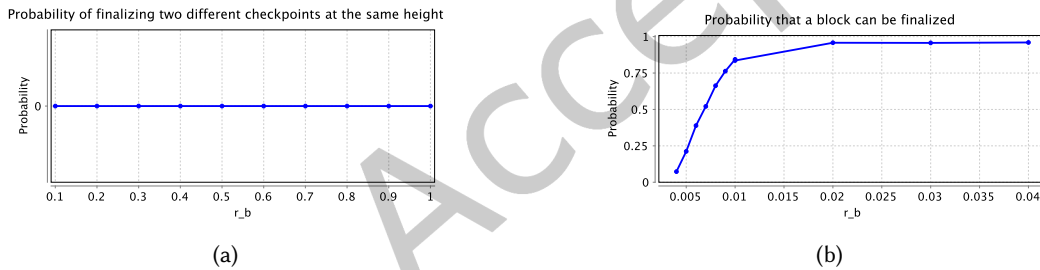


Fig. 3. Safety and liveness properties.

8 HYBRID CASPER STRESS TESTS AND ITS ROBUSTNESS TO ATTACKS

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_i 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. Finally, we study the robustness of Hybrid Casper to two attacks that have been studied in the literature [18, 27]. 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. In particular, regarding the attacks, we model them using the PRISM+ language and study their impact on the correct execution of the protocol through the verification capabilities of the model checker.

Hereafter, we will assume that all the mR_i are equal and we generically denote them with mR .

8.1 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 $mR = 1/14, 1/8, 1/7, 1/6$, respectively, i.e. blocks are created within 14, 8, 7 and 6 seconds, and the length of epochs is 64.

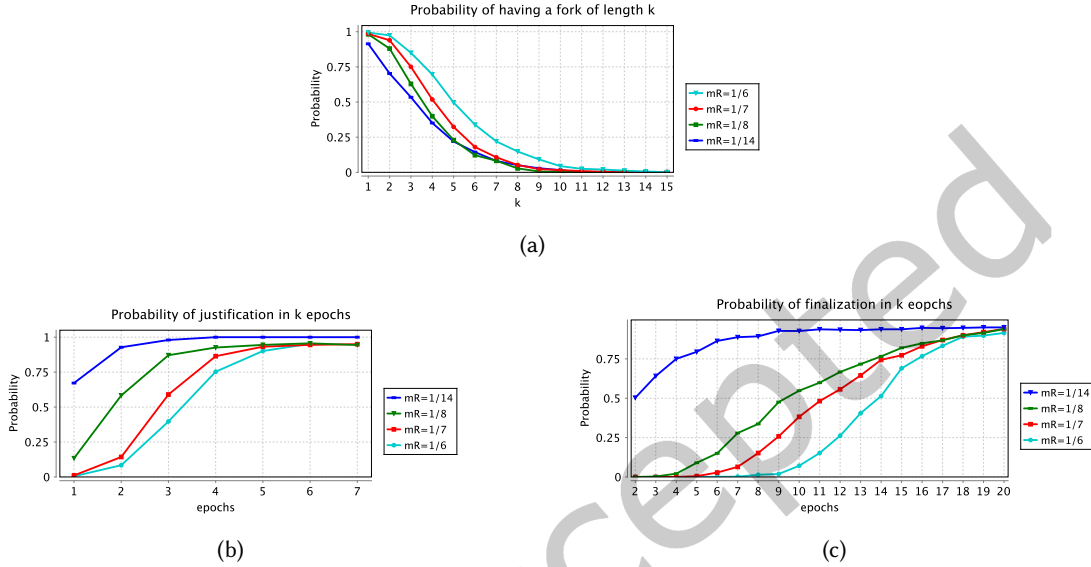


Fig. 4. Stress tests with different rates.

The results in Figure 4a show that the probability of a fork of length k for $mR = 1/8, 1/7, 1/6$ is higher than the one for $mR = 1/14$ (we run the analyses by considering $k \cdot 1/mR$ as bound time). In particular, the probability of a fork of length 7, i.e., $P_{7_{fork}}$, is 0.25 for $mR = 1/6$; 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 $mR = 1/8, 1/7, 1/6$, a block must wait for more epochs to be justified, see Figure 4c. In particular, with a creation rate $mR \leq 1/7$, a block cannot be justified within one epoch, but after two epochs, the probability rapidly increases. Even when $mR \leq 1/8$ the probability of justifying a block within 1 epoch is lower since the analysis shows that $P_{1_{just}} = 0.134$. Figure 4c reports the probability $P_{k_{fin}}$ of finalization 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 finalization within 2 epochs. This is due to the fact that also the probability of justification is very low. Moreover, with $mR = 1/6$, $P_{k_{fin}}$ starts to increase after 9 epochs, becoming 0.915 after 20 epochs.

8.2 Variation of the penalties

We analyse how the stakes change when the penalties for misbehaving validators vary. In Hybrid Casper, to incentivise correct behaviours, a bonus is given to validators when they vote checkpoints that are finalized; 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 she 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 $stake_j(i)$

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

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

that increases or decreases stakes according to a parameter γ . In our experiments (Figure 5) we consider three values for γ : 20%, 30% and 40%. In Figure 5a, we report how the stake of the *malicious* validator varies while

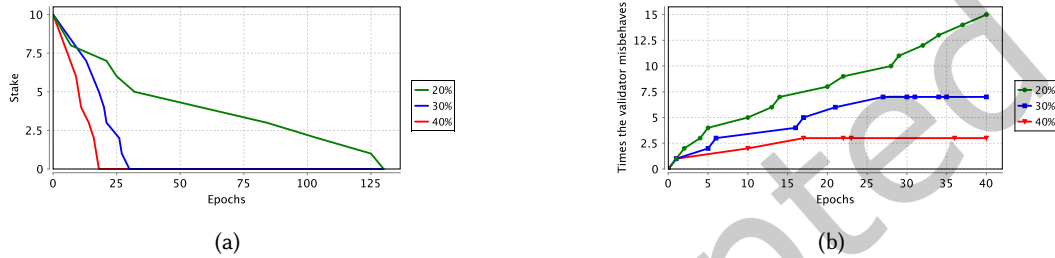


Fig. 5. Analyses with different penalties.

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%). The number of times a validator misbehaves with respect to the epochs is reported in Figure 5b. The higher is the penalty, the lower is 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 6 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 the reader can observe, at epoch 16 the stake of the validator

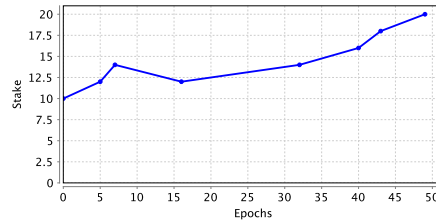


Fig. 6. How changes the stake of a validator.

decreases, that is due to the fact that it voted for the wrong checkpoint. The same stake increases because the validator voted correctly afterwards.

8.3 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 [18, 28]. The underlying idea of this attack is that an adversary control 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, she 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 5 – the new code is in Listing 8 – and we use the code in Listing 7 for the attacker. The attacker collects the blocks created by the victims in the set `setAttack_i`. She 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. Thus, in the code of the Network we have modified the sets the validators and the victims can send blocks to (lines 8-9 of Listing 8). 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 15 of Listing 7), the attacker makes them available to the rest of the network (line 24 of Listing 7). 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 `eclipseAttack_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 36-38). 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 12-15 of Listing 8). 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

$$P = \{ [F \leq T \quad \text{"eclipseAttack"}] \}$$

where `eclipseAttack` means that at least one of the `miner_i` boolean variables `eclipseAttack_i` is true.

Figure 7a 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 one, then the probability of a successful attack is much lower. In particular, when $mR \leq 1/8$, 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 7b 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 7a, that is the attacker does not succeed when $mR \leq 1/8$.

8.4 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 finalization and justification, since checkpoints can now receive the majority of the votes. Figure 8 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 4 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

```

1  module Attacker_i
2    STATE_i : [Start, Create, Receive, Move, Vote, Check, Fin, Comm] init Start;
3    setAttack_i : set [];
4    attack : bool init true;
5    nBlocksAttack : [0..1000] init 0;
6    nBlocks : [0..1000] init 0;
7    eclipseAttack_i : bool init false;
8    ...
9
10
11  [] (STATE_i=Start) ->
12    mR_i : (nBlocksAttack =nBlocksAttack+1)&(b_i'=createB(Validator_i,c_i,L_i))
13    &(c_i'=c_i+1)&(STATE_i'=Create);
14  [] (STATE_i=Start)&(nBlocksAttack>nBlocks)&!isEmpty(setAttack_i) ->
15    1 : (attack =false)&(STATE_i'=Comm)&(b_i'=receive(setAttack_i));
16  ...
17  [addB_i] (STATE_i=Create)&!isCP(b_i)&(attack=true) ->
18    1 : setAttack_i'=add(setAttack_i,b_i)&(L_i'=addBtoL(L_i,b_i))
19    &(STATE_i'=Start);
20  [addB_i] (STATE_i=Create)&isCP(b_i)&(attack=true) ->
21    1 : setAttack_i'=add(setAttack_i,b_i)&(L_i'=addBtoL(L_i,b_i))
22    &(STATE_i'=Vote);
23  ...
24  [communicate] (STATE_i=Comm) -> (setAttack_i '=remove(setAttack_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 : (nBlocksAttack '=nBlocksAttack+1)&(setAttack_i'=add(setAttack_i,b_i))
27    &(L_i'=addBtoL(L_i,b_i))&(STATE_i'=Vote);
28  [removeB_i] (STATE_i=Move)&canBeIns(L_i,b_i)&!isCP(b_i)&fromVictim(b_i) ->
29    1 : (nBlocksAttack '=nBlocksAttack+1)&(setAttack_i'=add(setAttack_i,b_i))
30    &(L_i'=addBtoL(L_i,b_i))&(STATE_i'=Start);
31  [removeB_i] (STATE_i=Move)&canBeIns(L_i,b_i)&isCP(b_i)&!fromVictim(b_i) ->
32    1 : (nBlocks '=nBlocks+1)&(L_i'=addBtoL(L_i,b_i))&(STATE_i'=Vote);
33  [removeB_i] (STATE_i=Move)&canBeIns(L_i,b_i)&!isCP(b_i)&!fromVictim(b_i) ->
34    1 : (nBlocks '=nBlocks+1)&(L_i'=addBtoL(L_i,b_i))&(STATE_i'=Start);
35  ...
36  [] (STATE_i=Check)&isJust(b_i,Votes,Stakes)&lastJ_i=lastboCP(b_i,L_i)&fromVictim(b_i) ->
37    1 : (lastJ_i'=b_i)&(L_i'= updateHF(L_i,lastJ_i))
38    &(STATE_i'=Fin)&(eclipseAttack_i'=true);
39  ...
40  endmodule

```

Listing 7. Pseudocode of an Attacker.

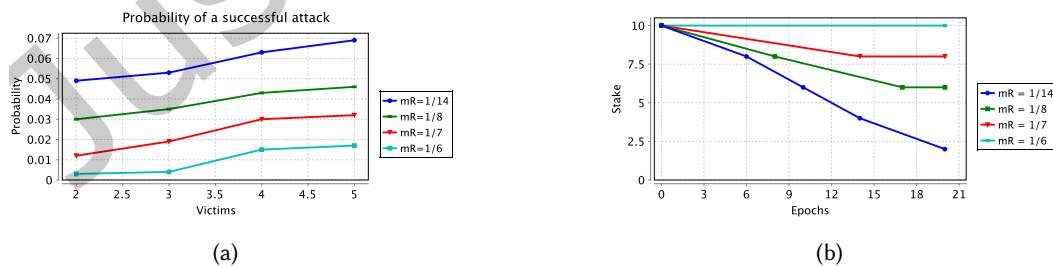


Fig. 7. Analysis in presence of an Eclipse attack.

```

1  module Network
2    for i from 0 to N:
3      SetB_i : set [];
4      N_i : set [];
5      Victims : set [];
6
7    [communicate] -> 1 : foreach k in N_i \ Victims{ SetB_k' = add(SetB_k, b_i); }
8    foreach i in Victims:
9      [addB_i] (attack = true) -> r_b: foreach k in Victims{ SetB_k' = add(SetB_k, b_i); }
10   foreach i N_i \ Victims:
11     [addB_i] (attack = true) -> r_b: foreach k in N_i \ Victims{ SetB_k' = add(SetB_k, b_i); }
12   for i from 0 to N:
13     [addB_i] (attack = false) -> r_b: foreach k in N_i{ SetB_k' = add(SetB_k, b_i); }
14   for i from 0 to N:
15     [removeB_i] -> 1 : SetB_i' = remove(SetB_i, b_i);
16  endmodule

```

Listing 8. Pseudocode of the Network in presence of an Attacker.

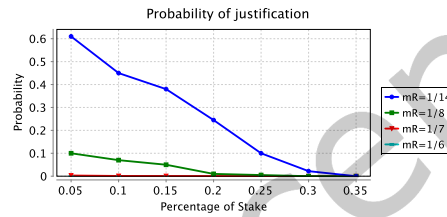


Fig. 8. Analysis in presence of a majority attack.

(and finalize) 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.

9 RELATED WORK

The blockchain was introduced by Haber and Stornetta [26] and only in the last few years, because of Bitcoin, the problem of analyzing the properties of the consensus protocols and the consistency of the ledgers managed by these protocols has attracted the interest of several researchers. Proof of work systems have been largely analyzed to ensure the correctness of the protocol [23, 42]. For example, Gervais et al. [24] introduce a quantitative framework to compare PoW blockchains. Their framework is based on Markov Decision Process and focuses on studying double-spending and selfish mining attacks. In contrast, our work considers Hybrid Casper which is a PoS consensus algorithm. Moreover, we model the system through CTMC where the participants of the protocol are expressed as modules in the PRISM language.

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 Hybrid Casper and then those contributions about other PoS protocols.

The initial contribution of Hybrid Casper by Buterin et al. [14] 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. As regards safety and liveness properties demonstrated by Buterin et al. [14], we have confirmed them even by changing the communication delay.

Safety of Casper has been formally proved using the Isabelle proof assistant [39] and in the Coq theorem prover [41] 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 [2]. 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 can be proved in a probabilistic version.

Coq has been also used to verify the Algorand PoS protocol [25]. 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. We think that the same comments above apply to the analysis of Algorand, once the protocol has been modelled in PRISM+.

With regards to other PoS protocols, another protocol that has been formally specified in a process calculus is Tendermint [35, 37, 44]. In this case, similarly to Section 6, the network of validators is modelled as a parallel composition of processes and the system is verified by means the PAT model checker [45], which is neither stochastic nor probabilistic. The authors prove that the consensus protocol is deadlock free and can reach consensus when at least 2/3 of the validators are in agreement. Our technique, by using a stochastic model checker, allows us to prove quantitative properties (*e.g.* probabilities of justification or finalizations) rather than qualitative ones.

PoS based blockchain design has been also studied by Bentov et al. [7, 8], both alone and in conjunction with PoW. Although they showed that the protocols are secure against some classes of attacks, no formal proof has been provided. A PoS protocol with rigorous security guarantees is Ouroboros [29], 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.

10 CONCLUSIONS

We have analyzed Hybrid Casper that will be adopted by Ethereum in the near future and that combines both PoW and PoS. We analyzed the protocol by means of PRISM+, an extension of the probabilistic model checker

PRISM with primitives for expressing the data types ledger and block, and the operations upon them (we used a similar technique to analyze the Bitcoin protocol [9]).

Once our model has been proved coherent with respect to the results by Buterin et al. [13], 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 showed that increasing the rate of creation severely impacts on the justification/finalization 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 attack* and the *majority attack*. Our results confirm that the protocol is robust against these two attacks when the original Hybrid Casper parameters are considered. We remark that PRISM+ is available online [22] and can be used to model and study quantitative properties of generic PoW and PoS protocols.

Our current PRISM+ model renders validators as pure stochastic processes and abstracts away the fact that they are actually rationale agents that compete each other to maximize a given utility function. Therefore, their behaviour is not only driven by the protocol but also by a given strategy that they follow when finalizing 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 formalization of utility functions [31] and will require a 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 Casper protocol. More specifically, we plan to study the *random validation mechanism*, which Casper will use to select the validator who can propose a new block [12]. 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 in 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 [25] and Tendermint [37] and quantitatively compare them to (Hybrid) Casper.

REFERENCES

- [1] Mansoor Ahmed and Kari Kostiainen. 2018. Don't Mine, Wait in Line: Fair and Efficient Blockchain Consensus with Robust Round Robin. *arXiv: Cryptography and Security* (2018).
- [2] Musab A. Alturki, Elaine Li, Daejun Park, Brandon Moore, Karl Palmkog, Lucas Peña, and G. Rosu. 2020. *Verifying Casper with Dynamic Validator Sets in Coq*. Technical Report.
- [3] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert K. Brayton. 1996. Verifying Continuous Time Markov Chains (*Lecture Notes in Computer Science, Vol. 1102*). Springer, Computer Aided Verification, 8th International Conference, CAV '96. , 269–276. https://doi.org/10.1007/3-540-61474-5_75
- [4] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. 2000. Model Checking Continuous-Time Markov Chains by Transient Analysis. In *Computer Aided Verification*, E. Allen Emerson and Aravinda Prasad Sistla (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 358–372.
- [5] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. 2003. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering* 29, 6 (2003), 524–541. <https://doi.org/10.1109/TSE.2003.1205180>

- [6] Stylianos Basagiannis, Sophia G. Petridou, Nikolaos Alexiou, Georgios I. Papadimitriou, and Panagiotis Katsaros. 2011. Quantitative analysis of a certified e-mail protocol in mobile environments: A probabilistic model checking approach. *Comput. Secur.* 30, 4 (2011), 257–272.
- [7] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. 2016. Cryptocurrencies Without Proof of Work. In *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 9604)*, Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff (Eds.). Springer, 142–157. https://doi.org/10.1007/978-3-662-53357-4_10
- [8] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. 2014. Proof of Activity: Extending Bitcoin’s Proof of Work via Proof of Stake [Extended Abstract]. *SIGMETRICS Perform. Evaluation Rev.* 42, 3 (2014), 34–37. <https://doi.org/10.1145/2695533.2695545>
- [9] Stefano Bistarelli, Rocco De Nicola, Letterio Galletta, Cosimo Laneve, Ivan Mercanti, and Adele Veschetti. 2021. Stochastic modeling and analysis of the bitcoin protocol in the presence of block communication delays. *Concurrency and Computation: Practice and Experience* (2021), e6749. <https://doi.org/10.1002/cpe.6749>
- [10] Stefano Bistarelli, Ivan Mercanti, Paolo Santancini, and Francesco Santini. 2019. End-to-End Voting with Non-Permissioned and Permissioned Ledgers. *J. Grid Comput.* 17, 1 (2019), 97–118. <https://doi.org/10.1007/s10723-019-09478-y>
- [11] Vitalik Buterin. 2013. Ethereum White Paper. <https://github.com/ethereum/wiki/wiki/White-Paper>.
- [12] Vitalik Buterin and V. Griffith. 2017. Casper the Friendly Finality Gadget. *ArXiv abs/1710.09437* (2017).
- [13] Vitalik Buterin, Diego Hernandez, Thor Kamphofner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X. Zhang. 2020. Combining GHOST and Casper. *CoRR abs/2003.03052* (2020).
- [14] Vitalik Buterin, Daniël Reijnders, Stefanos Leonardos, and Georgios Piliouras. 2020. Incentives in Ethereum’s hybrid Casper protocol. *International Journal of Network Management* 30, 5 (2020), e2098.
- [15] Huashan Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu. 2020. A Survey on Ethereum Systems Security: Vulnerabilities, Attacks, and Defenses. *ACM Comput. Surv.* 53, 3, Article 67 (jun 2020), 43 pages. <https://doi.org/10.1145/3391195>
- [16] Phil Daian, Rafael Pass, and Elaine Shi. 2019. Snow White: Robustly Reconfigurable Consensus and Applications to Provably Secure Proof of Stake. In *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 11598)*, Ian Goldberg and Tyler Moore (Eds.). Springer, 23–41. https://doi.org/10.1007/978-3-030-32101-7_2
- [17] Christian Decker and Roger Wattenhofer. 2013. 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, 1–10. <https://doi.org/10.1109/P2P.2013.6688704>
- [18] Evangelos Deirmentzoglou, Georgios Papakyriakopoulos, and Constantinos Patsakis. 2019. A Survey on Long-Range Attacks for Proof of Stake Protocols. *IEEE Access* 7 (2019), 28712–28725. <https://doi.org/10.1109/ACCESS.2019.2901858>
- [19] Giorgio Delzanno, Michele Tatarak, and Riccardo Traverso. 2014. 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 (EPTCS, Vol. 161)*, Adriano Peron and Carla Piazza (Eds.). 131–146. <https://doi.org/10.4204/EPTCS.161.13>
- [20] Carlos Faria and Miguel Correia. 2019. BlockSim: Blockchain Simulator. In *2019 IEEE International Conference on Blockchain (Blockchain)*. 439–446.
- [21] Cambridge Center for Alternative Finance. 2021. Cambridge Bitcoin Electricity Consumption Index. <https://cbeci.org/>. (last access 2021).
- [22] Letterio Galletta, Cosimo Laneve, Ivan Mercanti, and Adele Veschetti. 2022. PRISM+ software package, supporting material, and additional experiments. <https://github.com/adeleveschetti/casper-analysis>
- [23] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The Bitcoin Backbone Protocol: Analysis and Applications (*Lecture Notes in Computer Science, Vol. 9057*). Springer, Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques. , 281–310. https://doi.org/10.1007/978-3-662-46803-6_10
- [24] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. 2016. On the Security and Performance of Proof of Work Blockchains. *IACR Cryptol. ePrint Arch.* (2016), 555. <http://eprint.iacr.org/2016/555>
- [25] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *SOSP*. ACM, 51–68.
- [26] Stuart Haber and W. Scott Stornetta. 1990. How to Time-Stamp a Digital Document (*Lecture Notes in Computer Science, Vol. 537*). Springer, CRYPTO. , 437–455. https://doi.org/10.1007/3-540-38424-3_32
- [27] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse Attacks on Bitcoin’s Peer-to-Peer Network. In *Proceedings of the 24th USENIX Conference on Security Symposium* (Washington, D.C.) (SEC’15). USENIX Association, USA, 129–144.
- [28] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse attacks on {Bitcoin’s} {peer-to-peer} network. In *24th USENIX Security Symposium (USENIX Security 15)*. 129–144.
- [29] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynkov. 2017. 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 (Lecture Notes in Computer Science, Vol. 10401)*, Jonathan Katz and Hovav Shacham (Eds.).

- Springer, 357–388. https://doi.org/10.1007/978-3-319-63688-7_12
- [30] Marta Kwiatkowska, Gethin Norman, and David Parker. 2008. Analysis of a Gossip Protocol in PRISM. *SIGMETRICS Perform. Eval. Rev.* 36, 3 (Nov. 2008), 17–22. <https://doi.org/10.1145/1481506.1481511>
- [31] Marta Kwiatkowska, David Parker, and Clemens Wiltsche. 2018. PRISM-Games: Verification and Strategy Synthesis for Stochastic Multi-Player Games with Multiple Objectives. 20, 2 (April 2018), 195–210. <https://doi.org/10.1007/s10009-017-0476-z>
- [32] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2002. Probabilistic Symbolic Model Checking with PRISM: A Hybrid Approach. 2280 (2002), 52–66. https://doi.org/10.1007/3-540-46002-0_5
- [33] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems (*Lecture Notes in Computer Science, Vol. 6806*). Springer, Computer Aided Verification - 23rd International Conference, CAV 2011. , 585–591. https://doi.org/10.1007/978-3-642-22110-1_47
- [34] Marta Z. Kwiatkowska, Gethin Norman, and Roberto Segala. 2001. 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 (Lecture Notes in Computer Science, Vol. 2102)*, Gérard Berry, Hubert Comon, and Alain Finkel (Eds.). Springer, 194–206. https://doi.org/10.1007/3-540-44585-4_17
- [35] Jae Kwon. 2014. Tendermint : Consensus without Mining.
- [36] Leslie Lamport. 2006. Fast Paxos. *Distributed Comput.* 19, 2 (2006), 79–103. <https://doi.org/10.1007/s00446-006-0005-x>
- [37] Wai Yan Maung Maung Thin, Naipeng Dong, Guangdong Bai, and Jin Song Dong. 2018. Formal Analysis of a Proof-of-Stake Blockchain. In *2018 23rd International Conference on Engineering of Complex Computer Systems (ICECCS)*. 197–200. <https://doi.org/10.1109/ICECCS2018.2018.00031>
- [38] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. <https://bitcoin.org/bitcoin.pdf>.
- [39] R. Nakamura, T. Jimba, and D. Harz. 2019. Refinement and Verification of CBC Casper. In *2019 Crypto Valley Conference on Blockchain Technology (CVCBT)*. 26–38. <https://doi.org/10.1109/CVCBT.2019.00008>
- [40] E. Napoletano and John Schmidt. 2021. Decentralized Finance Is Building A New Financial System. <https://www.forbes.com/advisor/investing/defi-decentralized-finance/>. (last access 2021).
- [41] Karl Palmkog, Milos Gligoric, Lucas Peña, Brandon M. Moore, and G. Rosu. 2018. Verification of Casper in the Coq Proof Assistant.
- [42] Rafael Pass, Lior Seeman, and Abhi Shelat. 2017. Analysis of the Blockchain Protocol in Asynchronous Networks (*Lecture Notes in Computer Science, Vol. 10211*). Springer, EUROCRYPT. , 643–673. https://doi.org/10.1007/978-3-319-56614-6_22
- [43] Remigijus Paulavicius, Saulius Grigaitis, and Ernestas Filatovas. 2021. A Systematic Review and Empirical Analysis of Blockchain Simulators. *IEEE Access* 9 (2021), 38010–38028. <https://doi.org/10.1109/ACCESS.2021.3063324>
- [44] Jun Sun, Yang Liu, Jin Song Dong, and Chunqing Chen. 2009. Integrating Specification and Programs for System Modeling and Verification. In *2009 Third IEEE International Symposium on Theoretical Aspects of Software Engineering*. 127–135. <https://doi.org/10.1109/TASE.2009.32>
- [45] Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. 2009. PAT: Towards Flexible Verification under Fairness. In *Computer Aided Verification, Ahmed Bouajjani and Oded Maler (Eds.)*. Springer Berlin Heidelberg, Berlin, Heidelberg, 709–714.
- [46] Tatsuhiro Tsuchiya and Andre Schiper. 2007. Model Checking of Consensus Algorit. In *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*. 137–148. <https://doi.org/10.1109/SRDS.2007.20>

A NOTATION AND SYMBOLS

A summary of the symbols		
Symbol	Value	Meaning
r_b	1/12.6	communication latency of the network
mR_i	1/14 (1/8, 1/7, 1/6)	time needed to create a new block
hR_i	$1 - mR_i$	time needed to not create a block
rC_i	$1 / (len_{epoch} \times s)$	how often a validator should check for new justified/finalized blocks
N	13	number of validators
γ	20%, 30%, 40%	percentage of the increased/decreased stake
len_{epoch}	64	epochs length
P_{k_fin}	N/A	probability that a checkpoint is finalized within k epochs
P_{k_just}	N/A	probability that a checkpoint is justified within k epochs

Table 1. A table summarising the symbols used throughout the article.