Computer Communications 236 (2025) 108123

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

A stochastic analysis of the Gasper protocol☆

Cosimo Laneve [a], Adele Veschetti [b],*

[a] Department of Computer Science and Engineering, Mura Anteo Zamboni 7, Bologna, 40126, Italy
[b] Department of Computer Science, Hochschulstraße 10, Darmstadt, 64289, Germany

ARTICLE INFO

Keywords:
Stochastic modeling and analysis
Proof of stake
Blockchain fork

ABSTRACT

Ethereum has recently switched to a Proof of Stake consensus protocol called Gasper. We analyze Gasper using PRISM+, an extension of the probabilistic model checker PRISM with primitives for modeling blockchain data types. PRISM+ is therefore used to rapidly and automatically analyze the robustness of Gasper when tuning, up or down, several basic parameters of the protocol, such as network latencies and number of validators. We also study the effectiveness of Gasper in updating stakes and its resilience to three attacks: the balance, bouncing and time attacks.

1. Introduction

Ethereum is a network of peer-to-peer nodes that maintains a distributed ledger globally recording the occurrence of certain events. Due to the inherent asynchrony of the network, the main difficulty is the consistency of the ledger upon updates performed by different nodes. To overcome this problem, Ethereum relies on consensus protocols that impose a total order on the updates, which are called *blocks*. Historically, in line with Nakamoto's seminal work [1], these protocols have relied on a probabilistic mechanism known as Proof of Work (PoW). In this framework, nodes can update the ledger only upon solving a computationally demanding problem, which is a notable drawback because it causes a substantial waste of resources and energy [2], thus limiting the application of the technology (especially in contexts such as *pervasive computing* [3] where energy consumption is a critical issue). For this reason, the network recently underwent a significant protocol transformation, transitioning from a PoW model to a Proof of Stake (PoS) model, which introduced the Gasper consensus protocol, where some nodes are randomly designated to propose and verify new blocks. In particular, in Gasper, a subset of nodes of the network that own a *stake* – the *validators* – express votes for certain blocks – the *checkpoints*. These checkpoints pass through two stages: the first when the checkpoint is *justified*, which means that it has received at least two-thirds 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 blocks in the path from the corresponding checkpoint to the root of the ledger – the *blockchain*.

Since Gasper is pretty recent, the protocol may have corner cases that can pave the way to possible attacks and, up-to our knowledge, very few studies have already addressed its correctness [4,5]. We contribute to this issue by analyzing Gasper through a formal automatic verification technique that allows us (i) to predict how it behaves in different settings of the parameters and (ii) to understand its resilience and robustness to attacks.

Following [6,7], the automatic verifier we use is PRISM+, an extension of the stochastic model checker PRISM [8] with primitives for blockchain protocols. Gasper is rendered in PRISM+ as a parallel composition of 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. By tuning up and down these rates, it has been possible to analyze different settings of the basic parameters of the protocol rapidly and automatically and check the corresponding correctness.

The main contributions of this article are the following:

- We study the probability of justifying and finalizing checkpoints and the time needed to create a new block, thus providing empirical evidence of Gasper's robustness. In particular, we show that (i) the probability of justifying a checkpoint within 64 slots is 0.552 and it is greater than 0.9 in 96 slots and (ii) the probability of finalizing a checkpoint within 96 slots is almost 1.
- We test the protocol with respect to different time delays and verify that the probability of justification and finalization drops significantly by increasing the time needed for a block to be delivered. In particular, if the average delay time is 38 s, the

☆ This research has been supported by the SERICS project (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU and by the ATHENE project "Model-centric Deductive Verification of Smart Contracts".

* Corresponding author.
E-mail address: adele.veschetti@tu-darmstadt.de (A. Veschetti).

https://doi.org/10.1016/j.comcom.2025.108123
Received 3 July 2024; Received in revised form 21 November 2024; Accepted 25 February 2025
Available online 10 March 2025
0140-3664/© 2025 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

system is not able to justify a block with probability 1 within 9 epochs after its creation.

- We analyze the stakes' updates and we find out that, while in larger networks the rewards for proposing a block are bigger, in a smaller network it is more likely to become a proposer and receive a larger reward. This might reduce the interest in participating to the protocol, view the constant increase of validators in Ethereum.
- We evaluate the resilience of the protocol against the balance, bouncing and time attacks. We show that the protocol is highly affected by these attacks, therefore confirming the results in the literature [4,5], and analyze the effectiveness of patches that have been proposed.

This paper extends the analyses conducted in [9]. In particular, (*i*) we also evaluate how network latencies modify the process of justification and finalization; (*ii*) we study the trend of the average stake of the network when a single validator owns more stake than the other peers and when all the validators start with a greater amount of stake; (*iii*) we examine how the bouncing and the time attacks affect the probability of justifying/finalizing checkpoints.

The paper is organized as follows. Section 2 contains an overview of Gasper and Section 3 a quick introduction to PRISM and PRISM+. The PRISM+ model of Gasper is defined in Section 4 and the coherence of the model and the analyses of the security of the protocol are reported in Section 5. Section 6 compares our proposal with the literature and Section 7 draws some conclusions and discusses possible future work.

2. The gasper protocol

Gasper ledger is a tree of blocks with a pointer to a leaf block at maximal depth, called *handle*; the *blockchain* is the sequence of blocks from the handle to the root, called *genesis block*. Each block in the ledger has a height that is the length of the path from the block to the genesis block. Gasper consists of three main steps:

A. the selection of the validator that has to propose a new block, the creation of the block and its inclusion in the ledger;
B. the voting process of blocks and their irreversible storage in the blockchain, called finalization;
C. the incentive mechanism that rewards honest validators and punishes misbehaving ones.

We overview these steps in some detail in the following subsections.

2.1. Block creation

In Gasper, block creation involves *validators*, which are nodes chosen to create new blocks and validate transactions based on the number of coins they have stored as collateral in the network — the *stake*. This stake, which is at least 32 ETH (*Ethers*, the Ethereum cryptocurrency), allows the validator to be registered in the Gasper smart contract and to have a *unique index*. During a fixed time interval of 12 s, known as *slot*, a single validator is chosen to propose a block. If the designated proposer is offline during its slot, no new block will be generated for that slot and the other validators will have to wait until the end of the 12 s to proceed with a new block. Otherwise the proposer validator creates a block by collecting a number of transactions and a *block seed*, which combines the (hash of the) validator index and the epoch seed (see below). We recall that, even if the time frame in which the selected proposer can propose a block is limited to the current slot, it may happen that, due to the network latency, *one or more blocks may be received in the same slot*, which causes a *fork* in the ledger (*i.e.* multiple blocks at the same height in the ledger).

An *epoch* consists of 32 consecutive slots (therefore an epoch may contain a number of blocks that is less or equal to 32). The blocks that are at the beginning of an epoch are called *checkpoints* (also known as *epoch boundary blocks*).

The *validator selection process* is managed by an algorithm coded in the RANDAO smart contract [10]. This algorithm, which runs at the beginning of the epochs, uses the seeds that have been stored by the validators in the blocks at epoch e and combines these seeds with the epoch e-number by creating the *epoch seed* for epoch $e+1$. In turn, this value, is used by the RANDAO smart contract at epoch $e+1$ to generate a pseudo-casual sequence of 32 validator indexes with stake greater than 32 ETH that will propose the blocks at epoch $e+3$.

2.2. Finalization mechanism and the voting process

Finality refers to a critical property of checkpoints that makes them irreversible. When this happens, all the blocks in the blockchain with a smaller height are irreversible as well. To achieve finality, checkpoints undergo a two-step upgrade procedure. First, for a checkpoint to be *justified*, it must obtain approval from two-thirds of the total staked ETH, indicating a high level of confidence in its inclusion in the canonical chain. The second step involves *finalization*, which occurs when another checkpoint is justified on top of a justified block. This action commits to including the ancestor block in the canonical chain, making it irreversible.

The message (called *attestation*) containing a validator's vote for a checkpoint also contains a block vote, which is used to solve forks in the ledger. In particular, for every block vote, a weight is added to each block of the chain that has the voted block as a descendant. (The weight is proportional to the effective balance of the corresponding validator, see below.) Then, in case of forks, the so-called LMD-Ghost algorithm identifies the main chain by

1. considering the most recent attestation from each validator — the LMD (Latest Message Driven) process;
2. and selecting the chain with the heaviest subtree (the one with the most attestations) — the Ghost process.

2.3. Incentives and penalties

Validators earn rewards through various activities, including proposing blocks and making *consistent* votes with the majority of other validators in correspondence of checkpoints. In Gasper, a reward refers to the incentives distributed to participants (such as validators) for contributing to the maintenance, security, or functionality of the network. As explained in the documentation [11], the reward values for validator i are computed according to the so-called $\texttt{base_reward}_i$. This value represents the average reward that a validator would receive under optimal conditions per epoch and it is proportional to the *effective balance* (called $\texttt{stake}_i$ in our analyses) of the validator and inversely proportional to the number of validators on the network:

$$\texttt{base_reward}_i = \texttt{stake}_i \times 64/\Big(4 \times \sqrt{\sum_{j\in\mathbb{V}} \texttt{stake}_j}\Big).$$

The effective balance refers to the capped amount used for reward calculations, which maxes out at 32 ETH. This is distinct from the *actual balance*, which represents the total amount of ETH held by the validator. Thus, while the actual balance can fluctuate with the addition of rewards or penalties, only the effective balance is considered when determining rewards.

If a validator votes for the same block as the majority, it will receive a reward, which can be calculated as follows:[1]

$$\texttt{reward}_i = \frac{27}{32} \times \texttt{stake}_i \times \texttt{base_reward}_i$$

[1] In Gasper there are rewards within so-called *committees* as well, which use aggregators to get a consensus between components. Committees are groups of validators chosen to attest to blocks during each epoch, playing a crucial role in voting for block inclusion and contributing to block validation and finalization. We do not discuss this issue because we are assuming singleton committees.

The reward a proposer receives when one of the blocks it created has been validated is defined by the following formula, where n is the number of validators participating in the system:

$$\texttt{reward_bp}_i = \frac{n \times \texttt{reward}_i}{7 \times \texttt{stake}_i}.$$

Gasper also features a drastic penalty scenario, called *slashing*, which causes the expulsion of a validator from the network, together with the retrieval of its balance. Validators face slashing through two distinct scenarios: (*i*) proposing and endorsing two distinct blocks for the same slot and (*ii*) engaging in *double voting for checkpoints*. If these actions are detected, the validator is slashed: 1/32 of its balance is immediately burned and it is banned for 36 days; during this period additional penalties are also applied. For example, one such penalty is got when a validator does not send the attestation within the so-called *inclusion delay*, which is 32 slots (*i.e.* 384s). That is, votes beyond the same epoch of the checkpoint are not admitted and punished.

3. PRISM and PRISM+

The Gasper protocol is analyzed by means of a tool, PRISM+, which is an extension of PRISM. In this section we overview PRISM and PRISM+. We refer to [12] for a full account of PRISM.

PRISM [8] is a probabilistic model checker that computes the likelihood of the occurrence of certain events. The input of the model checker is a parallel composition of interacting modules, everyone representing 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. Every module is defined by a sequence of commands that specify how and under which conditions the module performs a transition and updates its internal state. In turn, commands have the pattern

```
[a] guard -> rho_1:update_1+...+rho_n:update_n;
```

where a, called *action* is used to synchronize commands of different modules (actions may be omitted, meaning that no synchronization is required); guard is a *predicate* over the state variables of the system (those of the module and of the other modules); update_i defines a list of *assignments* to its 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.

To illustrate PRISM semantics we discuss the code in Listing 1 (which is taken from the documentation[2]). This code defines an N-place queue of jobs and a server that removes jobs from the queue and processes them.

The module queue (lines 6–11) uses the variable q to represent the size of the queue (N defines the maximal capacity). Transitions specify the operations: The enqueue operation (line 8) inserts a new element with rate mu, if the queue is not full ($q < N$); the operation is modeled 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 command at line 9) says that no new element is inserted when the queue is full. The command at line 10 dequeue an element, and it is performed provided that the server can consume it. For this reason it has the action name serve that constrains server to perform a transition with the same name.

The module server (lines 13–17), uses s to define whether the server is busy or not. When $s = 0$ (the server is not busy), the command at line 15 lets the server synchronize with queue through the action serve. After this synchronization the server updates its state to busy

```
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       [] q<N -> mu:(q'=q+1);
9       [] q=N -> mu:(q'=q);
10      [serve] q>0 -> lambda:(q'=q-1);
11 endmodule
12
13 module server
14      s : [0..1];
15      [serve] s=0 -> 1:(s'=1);
16      [] s=1 -> gamma:(s'=0);
17 endmodule
18
19 label "full" = q = N;
```

Listing 1 An N-place queue and a server in PRISM

($s' = 1$). The rate of the synchronization is the product of the two individual rates (in this case, lambda * 1). The command at line 16 states that a busy server ($s = 1$) can complete its task with rate gamma. Line 19 defines a label. Labels are a way of identifying sets of states that are of particular interest for expressing properties (see below). In this case, it represents the states in which the queue q is full.

PRISM supports different kinds of probabilistic formalisms; in this paper we use Continuous Time Markov Chains (CTMC), which are transition systems (as the one above) where each transition is labeled by a positive real number, called *rate*. In particular, if the rate of a transition from a state S_0 to S_1 is r then the probability of moving from S_0 to S_1 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_0 in a given time. For example, in Gasper, the creation of blocks can be approximated by an exponential distribution with rate 1/12 (because a block is created every 12 s) and an exponential distribution also approximates the probability of delivering blocks across the network [13].

When a CTMC state has several exiting transitions (in the above example, when $0 < q \leq N$), then the probability of choosing one of them depends on their rates — this is known as *race condition*. For example, if $r_1, \cdots, r_n$ are the rates of transitions exiting from a state S_0 (and entering on pairwise different states), then the probability of taking a transition with rate r_i is r_i/R, where $R = r_1 + \cdots + r_n$. 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*. We use a basic property of CTMC models: 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.

In PRISM, properties of CTMC models are expressed in Continuous Stochastic Logic (CSL) [14–16], which is an extension of temporal logic with a probabilistic operator. 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.

In the example in Listing 1, the probability that the queue is full within 10 time units is denoted by the CSL formula

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

```
P=?[F<=10 "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 P_{apx} of the actual probability P for the formula ϕ, and gives a probabilistic guarantee on the accuracy of P_{apx}. In particular, P_{apx} 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*.

3.1. The Extension PRISM+

PRISM+ [17] extends PRISM by adding a native support for expressing and manipulating dynamic data types, such as ledgers, in order to provide a generic set of primitives that can be used to model and analyze different kinds of blockchain protocols.

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 analyze them by means of the statistical model checking engine that we have discussed above. These types and the corresponding operations are presented in the following paragraphs.

Blocks, noted b, p, …, are triples $(\mathtt{v}^{\mathtt{n}};\mathtt{p};\mathtt{h})$, where v is the *name* of the validator that created the block; n is a unique numeric label; p, called *father*, is the name of another block which $(\mathtt{v}^{\mathtt{n}};\mathtt{p};\mathtt{h})$ points to; h is the *height* of the block in the ledger. For instance, $(\mathtt{v}_3^0;\mathtt{v}_4^7;3)$ is a block named $\mathtt{v}_3^0$, which is the first block created by $\mathtt{v}_3$, whose father is the block named $\mathtt{v}_4^7$ and which is at height 3. The operations on blocks are:

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

Ledgers, noted L, L′, …, are tuples ⟨T; f; p; K⟩ 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 $\mathtt{K(b)} > \mathtt{K(b')}$. The root of T is called *genesis block* and is denoted by $(\mathtt{genesis}^0;\mathtt{genesis}^0;0)$. The *blockchain* of L is the sequence of blocks that starts from the handle and reaches the genesis.

The operations on ledgers are:

- canBeIns(L,b) returns true if b can be inserted in L (*i.e.* if the father of b is in L), false otherwise;
- addBtoL(L,b) inserts b in L and returns the updated ledger (precondition: canBeIns(L,b) = true); it also updates the mapping K and, in case, it updates the handle;
- lastCP(L) and lastboCP(L) return the last checkpoint and the last-but-one checkpoint in L, respectively;
- lastF(L) returns the last finalized block in L, *e.g.* lastF (⟨T, f, p, K⟩) = f.
- updateHF(L,b) takes a ledger $\mathtt{L} = \langle \mathtt{T};\mathtt{f};\mathtt{p};\mathtt{K}\rangle$ and a block b that has been finalized such that $\mathtt{K(b)} > \mathtt{K(f)}$ and returns $\mathtt{L'} = \langle \mathtt{T};\mathtt{b};\mathtt{p'};\mathtt{K}\rangle$ 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:

Let $\mathtt{L}_1 = \langle \mathtt{T}_1;\mathtt{f}_1;\mathtt{p}_1;\mathtt{K}_1\rangle$, …, $\mathtt{L}_n = \langle \mathtt{T}_n;\mathtt{f}_n;\mathtt{p}_n;\mathtt{K}_n\rangle$ be a set of ledgers and let m be the maximal height of the handles $\mathtt{p}_1, \ldots, \mathtt{p}_n$. Let also $\mathtt{L}_{i_1}$, …, $\mathtt{L}_{i_k}$ be the ledgers in the above set with the handle at height *m*. We say that the set $\mathtt{L}_1, \ldots, \mathtt{L}_n$ has a fork of length $m - h$, where h is the length of the maximal common suffix of the blockchains of $\mathtt{L}_{i_1}, \ldots, \mathtt{L}_{i_k}$.

The following operations are used to compute and verify forks:

- $\mathtt{calculateFork}(\mathtt{L}_1,\ldots,\mathtt{L}_n)$ returns the length of the fork in $\mathtt{L}_1,\ldots,\mathtt{L}_n$, 0 if there is no fork.
- $\mathtt{verifyCP}(\mathtt{L}_1,\ldots,\mathtt{L}_n)$ returns true if $\mathtt{lastCP}(\mathtt{L}_1) = \cdots = \mathtt{lastCP}(\mathtt{L}_n)$, false otherwise.

Sets, noted S, S′, …, are collections of blocks, whose operations are:

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

In our model, sets are used to store the blocks that have been received but have to be inserted in the local ledger.

Maps, noted H, H′, …, 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 in the Ethereum documentation [18,19];
- 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′.

4. The Gasper model in PRISM+

The current implementation of Gasper consists of several interacting modules. Our PRISM+ model closely adheres to it by defining Gasper as a parallel composition of different modules: Validators, Network, Vote_Manager, Randao, RandaoSelection and Global. Each module plays a critical role in simulating and analyzing the behavior of the system, contributing to the overall robustness and reliability of our implementation. The architecture of our model is in Fig. 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; Network implements the broadcast communication mechanism among validators; Global is an auxiliary module that computes the length of forks — see Section 5; $\mathtt{Validator}_i$ are the processes modeling the validators and Randao and RandaoSelection simulate the random selection of the proposers for the next epoch.

For clarity sake, we present a sugared version of the PRISM+ code of Gasper; the online repository [17] contains the actual implementation, the verified properties with the data of our analyses and the instructions for the use of the tool.

```
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     [] (STATE_i=Start)&(validatorID=i) ->
8          1 : (b_i'=createB(Validator_i,c_i,L_i))&(c_i'=c_i+1)
9          &(STATE_i'=Create)&(validatorID'=-1);
10    [] (STATE_i=Start)&((voteID_0=i)|(voteID_1=i)|(voteID_2=i)) -> 1 : (STATE_i'=Vote);
11    [] (STATE_i=Start) -> 1 : (STATE_i'=Receive);
12    [] (STATE_i=Start) -> rC_i : (b_i'=lastCP(L_i))&(STATE_i'=Check);
13    [addB_i] (STATE_i=Create) ->
14               1 : (L_i'=addBtoL(L_i,b_i))&(STATE_i'=Start);
15    [voteB_i] (STATE_i=Vote) -> 1 : (STATE_i'=Start);
16    [] (STATE_i=Receive)&!isEmpty(Set_i) ->
17               1 : (b_i'=receive(Set_i))&(STATE_i'=Move);
18    [] (STATE_i=Receive)&isEmpty(Set_i) -> 1 : (STATE_i'=Start);
19    [removeB_i] (STATE_i=Move)&canBeIns(L_i,b_i) ->
20               1 : (L_i'=addBtoL(L_i,b_i))&(STATE_i'=Start);
21    [] (STATE_i=Move) & !canBeIns(L_i,b_i) -> 1 : (STATE_i'=Start);
22    [] (STATE_i=Check)&isJust(b_i,Votes,Stakes)&lastJ_i=lastboCP(b_i,L_i) ->
23               1 : (lastJ_i'=b_i)&(L_i'= updateHF(L_i,lastJ_i))&(STATE_i'=Fin);
24    [] (STATE_i=Check)&isJust(b_i,Votes,Stakes)&lastJ_i!=lastboCP(L_i) ->
25               1 : (lastJ_i'=b_i)&(STATE_i'=Start);
26    [] (STATE_i=Check)&!isJust(b_i,Votes) -> 1 : STATE_i'=Start;
27    [finB_i] (STATE_i=Fin) -> 1 : STATE_i'=Start;
28 endmodule
```

Listing 2 The code of a `Validator`.

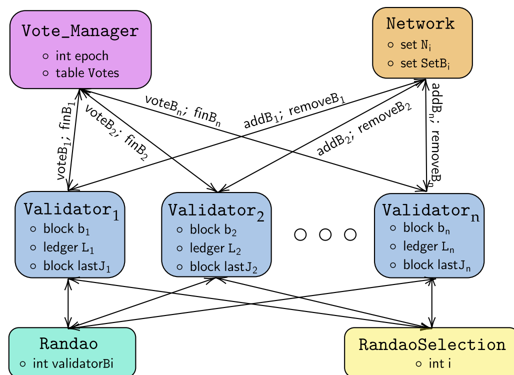

Fig. 1. The Ethereum PoS model architecture.

4.1. The `Validator`

The $\mathtt{Validator}_i$ module is reported in Listing 2. It is defined as a state machine with seven states, whose current one 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 `Start`, the validator may either (*i*) create a new block if it has been selected by the `RANDAO` algorithm (*c.f.* the value of the variable `validatorID`) and transit to the `Create` state (line 9), or (*ii*) has been selected to vote in this slot and transit to `Vote` (line 10), or (*iii*) receive a new block from the network and transit to `Receive` (line 11), or (*iv*) check whether a checkpoint can be justified and transit to `Check` (line 12), provided the block is the end of an epoch. The rate `rC_i` represents how often a validator should check for new justified/finalized blocks. In the Gasper protocol [20], in the absence of excessive delays or congestion, this happens within the slot to which they have been assigned. Thus $\mathtt{rC_i} = \mathtt{s}/(len_{epoch})$, where len_{epoch} is 384 and `s` is 12.

In the state `Create` (the validator creates a new block), the validator updates its ledger and sends the created block to `Network` (see action `addB_i` at lines 13–14) to forward it to the other validators. The rate of this action is determined by the companion action in `Network`, which expresses the communication latency of the network (*c.f.* line 6 of Listing 6).

If the new block is a checkpoint (the height of the block is a multiple of $len_{epoch}/\mathtt{s}$, *i.e.* 32 which represents the number of slots in an epoch), `Validator` returns to `Start`.

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 15).

In the state `Receive`, the validator verifies whether `Network` has blocks to deliver (lines from 16 to 18), and in case it transits to the state `Move`; otherwise, it returns to `Start`.

In `Move`, `Validator` verifies whether the block can be inserted in its own ledger (lines from 19–20). If this is the case, `Validator` adds the block and returns to the initial state (line 21). It is possible that a block that has been received cannot be inserted in the local ledger – these blocks are called *orphans* in the literature – because its entire ancestry (yet) is missing. In this case, the block is left in the local set `set_i` of `Network` and will be added afterwards.

In the state `Check`, the validator verifies whether the last checkpoint, say $\mathtt{C}_L$, has received the majority of the votes (i.e. $\mathtt{C}_L$ has been justified) and whether the last but one checkpoint in the blockchain of $\mathtt{C}_L$, say $\mathtt{C}_A$, is also justified. If this is the case, $\mathtt{C}_A$ becomes finalized and $\mathtt{C}_L$ becomes justified — in lines 22–23 this is performed by storing $\mathtt{C}_L$ 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.

If $\mathtt{C}_A$ is not justified then $\mathtt{C}_L$ is stored in `lastJ_i` (lines 24–25). In any case, the validator returns to `Start` and the process starts again.

```
1  module Randao
2      for i from 0 to 31:
3          ValidatorB_i: int init -1;
4          for j from 0 to 2:
5              Voter_ji: int init -1;
6      for i from 0 to N:
7          index_i : int init i;
8
9      [] (state=0)&(randao=true) -> 1 : (state′=1)& for i from 0 to N:
10                              if Stakes[i] < 32 : index_i′ = -1;
11     [] (state=1) ->
12             1: (randao′=false)&(state′=0)& for i from 0 to 31:
13                 ValidatorB_i′=randomNumber(epoch,i, index_0,...,index_N)
14                 & Voter_0i′=randomNumber(epoch,i+31, index_0,...,index_N)
15                 & Voter_1i′=randomNumber(epoch,i+63, index_0,...,index_N)
16                 & Voter_2i′=randomNumber(epoch,i+95, index_0,...,index_N)
17
18 endmodule
```

Listing 3 The code of Randao.

```
1  module Randao_Selection
2      i : [0..32] init 0;
3      [] (i=0)&(randao=false) -> 1 : (validatorID′=ValidatorB0)&(i′=i+1);
4      for j from 1 to 31:
5          [] (i=j) -> slot :
6              (validatorID′=ValidatorB_j)&(voterID_0′=Voter_0i)
7              &(voterID_1′=Voter_1i)&(voterID_2′=Voter_2i)&(i′=i+1);
8          [] (i>31) -> 1: (i′=0)&(randao′=true);
9  endmodule
```

Listing 4 The code of RandaoSelection.

4.2. *The* `RANDAO`

The `RANDAO` process has been split in two different modules: `Randao` and `RandaoSelection`.

The first one, reported in Listing 3, generates 32 random numbers that represent the index of the validators that will be allowed to create a block for each slot. The process also selects a number of validators that vote for the block (in Gasper this number is 64, in our simulations it is 3 — see lines 11–17 in Listing 3). For the pseudo-random selection of the validators, a function called `randomNumber` has been implemented which takes as input the indices of the active validators (the validators with at least 32 ETH), the epoch number and the number of the slot. The function returns the index of the next block proposer.

The code in Listing 4 is used to change the indeces of the selected proposer and validators for each slot. For each slot, index `i`, the global variables `validatorID` collects the extracted value for the corresponding slot, `voterID_0`, `voterID_1` and `voterID_2` store the extracted values for the slot.

4.3. *The* `Vote_Manager`

The `Vote_Manager` module is used to initialize, track and update the stakes of each validator belonging to the network.

The internal state of the `Vote_Manager` 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`. 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 (line 11) then no update occurs.

4.4. *The* `Network`

The `Network` module in Listing 6 manages the delivery of blocks to the validators and the addition or elimination of a block to the main blockchain, considering the addition or removal delays that are foreseen by the protocol used.

The transitions in this module manage the steps before justifying or finalizing a block. If a block needs to be added, in the `addB_i` transition, then it will be added to each validator's set, so that everyone can have the block in its set. The opposite situation exists in `removeB_i`, which is carried out only in the set of the user who created the block.

As regards the rate r_b, we set it to 1/12.6, which is the broadcast delay of the Bitcoin network [13], in order to have an average delay value that allows us to compare our simulation with other models. However, in Section 5, we will consider different latencies to test the protocol under different settings.

4.5. *Simplifications*

In our model and in the analyses presented in the next sections we overlook some details of Gasper that are not relevant to the properties

```
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 5 The code of Vote_Manager.

```
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] -> r_b : SetB_i'=remove(SetB_i,b_i);
9  endmodule
```

Listing 6 The code of Network.

we are interested in. In particular, since our goal is to study the behavior of the protocol to changes in basic parameters, such as the rate of creating new blocks and the percentages in the rewards/penalties system, we assume that:

1. blocks are black boxes: we omit any informations that is not relevant for the consensus, in particular Solidity transactions. In fact, the analysis of Solidity transactions would add complexities related to the computation time of every transaction. This issue is outside the scope of our analysis because it does not affect the basic consensus mechanisms we are studying;
2. the network consists of validators that also create new blocks. In particular, we drop the distinction between validators and proposers and the grouping of proposers in committees;
3. the selection of the proposers and the voters for blocks is defined by a randomNumber function, which does not consider the block seeds of the previous epoch (*c.f.* Section 2.1).

Finally, the dynamic networks, that is validators that may enter or exit the protocol (*c.f.* end of Section 2), have been overlooked in this work. The modeling and the corresponding analysis of this feature is left to future work.

5. Results

The section reports the experiments performed in order to test Gasper. Since the protocol is new, we have not found any benchmark in the literature to verify the coherence of our PRISM+ model. Therefore we decided to compare it with those about the previous Ethereum protocol — Hybrid Casper. Then, we will report a number of relevant security tests concerning the stake analysis and the results obtained in presence of three attacks: *balance*, *bouncing* and *time* attacks.

The experiments have been carried out on a Virtual Machine with 8 VCPU and 64 GB RAM. The stake analyses were conducted through simulations, considering 100,000 validators. The other experiments have been conducted by setting the PRISM+ model checker to generate 100000 samples of protocol executions. The verified systems were composed by n validators, with $n = 6, 8, 10, 13, 16$, and the experiments are always run until we observe a stabilization of results. Usually with networks larger than 10-13 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.

5.1. Coherence of the model — comparison with hybrid casper

Hybrid Casper [21] is a consensus protocol that has been used by Ethereum in order to ensure a smooth transition from the original PoW protocol to Gasper. This protocol keeps the ledgers consistent by using two 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 s. Hybrid Casper PoS mechanism for finalizing checkpoints is the same as in Gasper, with the difference that epochs' length consists of 64 blocks. Another relevant difference between the two protocols is that Gasper is strictly based on timeframes to propose and finalize blocks, while Hybrid Casper uses the height of blocks in the ledger to trigger the finalization process (and, in turn, blocks may take more than 14 s because the PoW may be longer). Therefore, we expect that Gasper has a better performance in proposing and finalizing blocks. The following analyses confirms this expectation.

The results obtained by analyzing Gasper (blue lines) are compared with those in the literature [22] whenever available (green line) and with those obtained for Hybrid Casper, reported in [7] (red line). As regards *block creation*, which specifically refers to the process of proposing a new block, Fig. 2 reports the process over a duration of 25 s (a bit more of two Gasper slots and a bit less of two Hybrid Casper slots).

We notice that a block is created in Gasper within 12 s with probability of 1. On the contrary, in Hybrid Casper the probability of creating a block within 14 s is around 0.9. This is actually reasonable, since a Proof of Work computational problem has to be solved and it is not evident that at least a node has been able to solve it in 14 s.

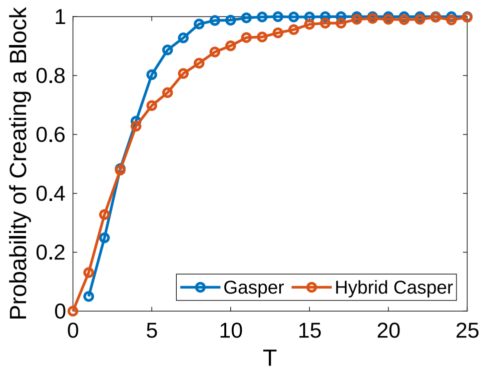

Fig. 2. Block creation probability over time.

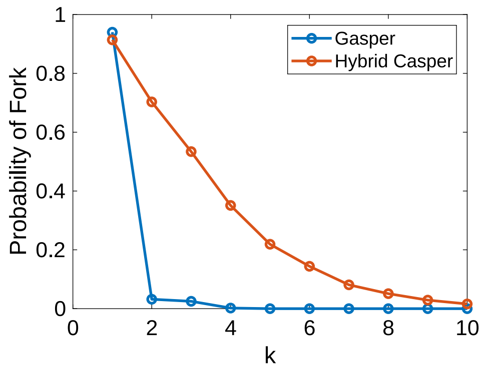

Fig. 3. Probability of a fork of length k.

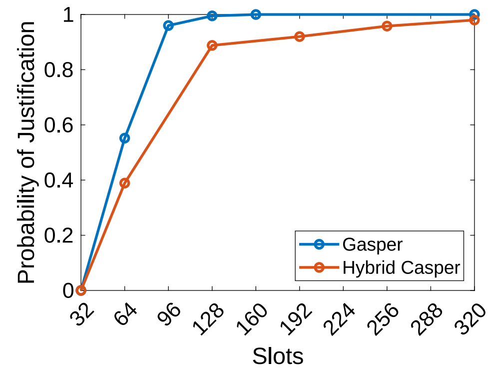

Fig. 4. Justification.

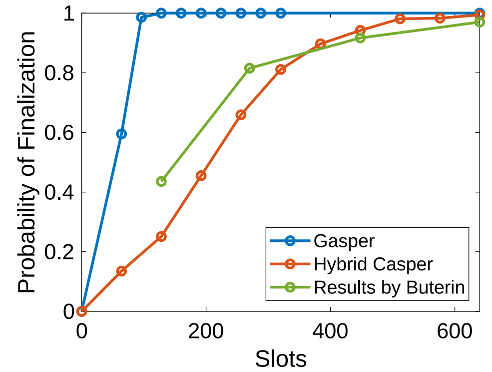

Fig. 5. Finalization.

As regards *forks*, Fig. 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 $k * 12$ 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 0.005 for forks of length 3. Comparing the results with the ones obtained for Hybrid Casper, it is evident that the probability is way lower. This is due to the fact that only one proposer is selected in the Gasper protocol, thus a fork of length 1 may happen due to the latency of the network, but longer forks are very unlikely.

Last, as regards *justification* and *finalization*, we compare the probabilities for Gasper and Hybrid Casper considering a network delay of 12.6 s. In Figs. 4 and 5, on the x-axis, we report the number of the slots we are considering, instead of the number of epochs because, in Hybrid Casper an epoch has a length of 64 blocks, while in Gasper an epoch is a period of 32 slots, in this way we can compare the results obtained for both the protocols at the same time.

It turns out that for Gasper the probability of justifying after 64 slots is 0.552. It is also worth to notice that, when the epochs are 5, the probability is greater than 0.98 which is in accordance with [21] where this probability is stated to be greater than 0.5 in one epoch. We note that, at slot 32, the probability of justification is zero because the length of an epoch in Gasper is defined as 64 slots. Since justifications only start after the completion of an epoch, no justification can take place until slot 64. Only at that time enough validator attestations could have been collected to meet the threshold required for justification.

In Fig. 5, we compare the probability of finalizing blocks also with the results presented in [21]. In Hybrid Casper, a block is created by solving a PoW computational puzzle and, thus, the time needed may vary and forks may be more probable. On the contrary, Gasper has a lower probability of forks (as shown in Fig. 3) and, thus, a higher probability of finalizing checkpoints. In particular, because of the process of creation of blocks, this probability is already greater than 0.6 after 32 slots.

5.2. Different network latencies

In order to analyze how the Gasper protocol is affected by the network latency, we report some analyses considering 4 different settings: no delay, a delay of 12.6 s, a delay of 25.2 s and a delay of 37.8 s. In our analysis we do not consider block creation since it is not influenced by network latencies. Indeed, block creation only relies on the randomness provided by the `RANDAO` smart contract, which ensures fair and unpredictable proposer selection. Fig. 6 reports the impact of different network latencies for justification and finalization. We notice that the more time is needed for a block to be received by the network, the less likely is for a checkpoint to be justified/finalized.

In particular, a delay of 38 s does not allow to the system to have a probability of 1 of justifying a checkpoint before 9 epochs after the creation. The same applies for the finalization, where we have a probability ≈ 0.99 only after 12 epochs, which would highly affects the security of the system.

5.3. Stake analysis

In this section we conduct an in-depth analysis of the average stakes of validators in the network. The purpose is to clarify the effectiveness of the Gasper protocol in the context of increasing or decreasing stakes during the epochs. In the experiments of Figs. 7–9, we consider a network of honest validators with probability 0.8 of sending correct attestations and 0.2 of sending incorrect ones. This 0.8/0.2 ratio has been chosen to analyze how variations in attestation accuracy impact validator stakes and to reveal the protocol's robustness in non-ideal conditions. The ratio was selected because, with an incorrect attestation

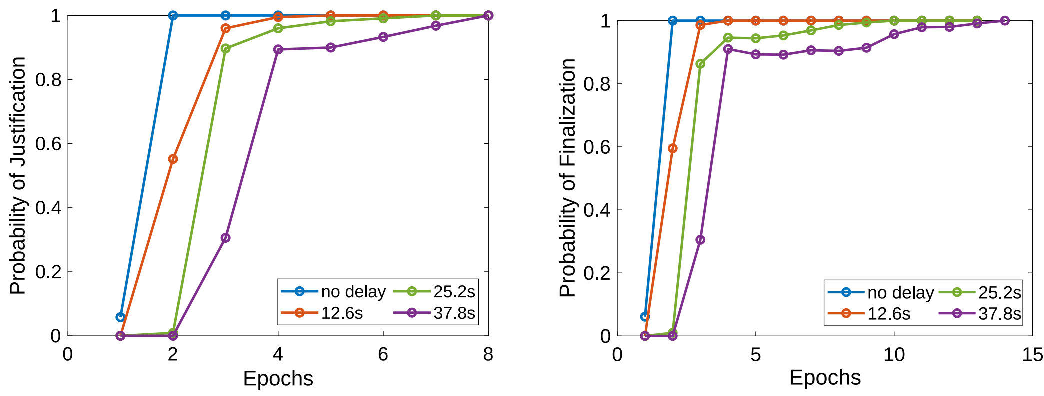

Fig. 6. Justification (left) and finalization (right) with different delays.

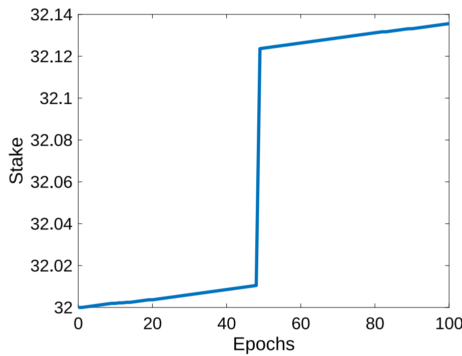

Fig. 7. $\texttt{act_stake}_i$ with 100k validators having equal initial stake.

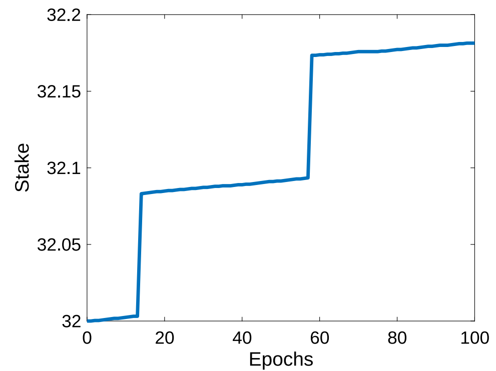

Fig. 8. $\texttt{act_stake}_i$ with 50k validators having equal initial stake.

probability of 0.1, the differences in the analysis would be too small to be observed. On the other hand, using a higher probability would be unrealistic and not reflective of typical network conditions. The simulations will not incorporate the more punitive aspects of Gasper, such as the inactivity leak and the slashing mechanisms. In particular, when computing the reward for the block proposal, a maximum reward is considered and applied according to the validators' votes are correct or not.

As regards the gas rewards and penalties, in the experiments we used the actual Ethereum protocol parameters described in Section 2.3. The experiments highlight the growth of the actual balance $\texttt{act_stake}_i$ of a validator i. This balance shows validator's total holdings over time and captures the volatility that may result from immediate changes in stake. It is different from the *effective balance* of Section 2.3 that was noted $\texttt{stake}_i$ and capped by 32 ETH.

Fig. 7 spots the actual balance of a validator in 100 epochs, assuming there are 10^5 validators with a stake of 32 ETH. On the x axis, we will indicate the time intervals in epochs, with the understanding that one slot equals 12 s and one epoch equals 32 slots (hence 384 s).

In this simulation, the base reward is 9050 Gwei (1 Gwei is equal to 10^{-9} ETH) and the reward for the proposer of the block is 0.113 ETH. We observe that, in Fig. 7, the validator obtains a significant growth in stake around epoch 50, which comes from having become a block proposer.

The next experiment was carried out to compare the previous results with a smaller network, in order to understand whether there is a regular trend or not. Fig. 8 reports the actual balance of a validator in 100 epochs, assuming there are 0.5×10^5 validators with a stake of 32 ETH.

In this simulation, the base reward is 12800 Gwei, so 0.080 ETH for the proposal of a block. The slopes in Fig. 8 indicate that the validator has became a block proposer twice in 100 epochs, around epochs 10 and 60, therefore exceeding the 32.12 ETH reached in the experiment with 10^5 validators. This means that, even if in a larger network the rewards for proposing a block are bigger, in a smaller network it is more likely to become a proposer and receive a larger reward. Therefore, the constant increase in the number of validators registered on the Ethereum network could lead to a loss of interest in participating to the protocol.

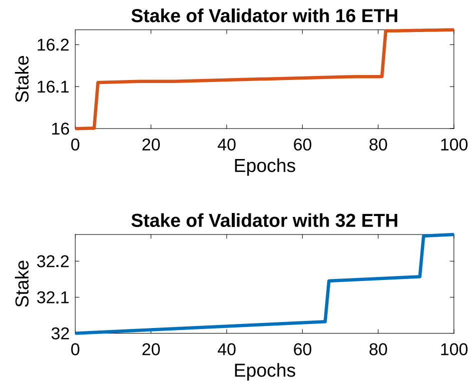

Fig. 9. $\texttt{act_stake}_i$ with 50k validators.

Fig. 9 shows the comparison between the stake of a validator with 32 ETH and the stake of a validator among others who have 16 ETH, within a network of 50000 validators, all holding 16 ETH except for one holding 32 ETH. The trend is always analyzed over 100 epochs, considering the base reward to be 18101 Gwei and the reward for the block proposal 0.113 ETH. This experiment highlights that the presence of a unique validator with a larger initial stake does not let it to enrich considerably

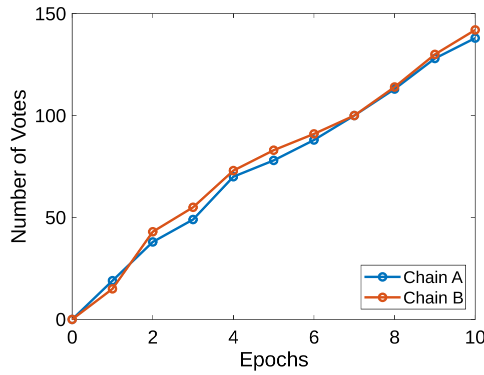

Fig. 10. Balance between votes during a balance attack.

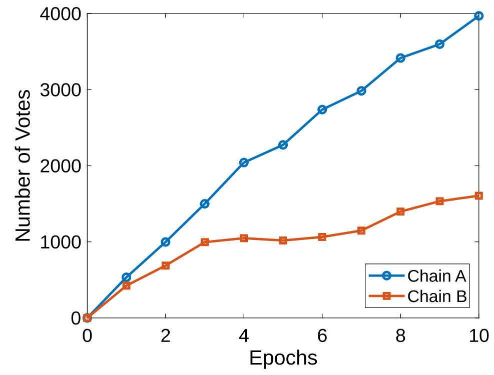

Fig. 11. Trend of the votes for the two chains A and B in 10 epochs.

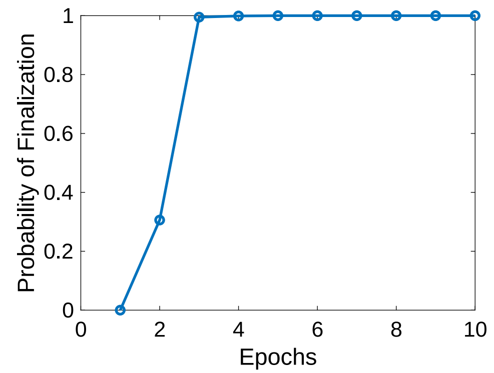

Fig. 12. The votes difference received between the two chains allows to establish liveness.

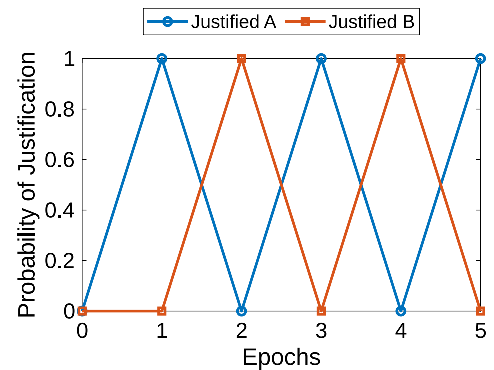

Fig. 13. Phenomenon of bouncing from chain A to chain B.

more than the others, given that the stake amount influencing the proposer's selection among validators is 32 ETH.

One noteworthy observation that emerged from our investigation was Gasper's ability to incentivize honest validators, fostering gradual stake growth over time. This growth of stakes follows a linear trajectory rather than experiencing an exponential ascent, regardless of their ETH holdings.

5.4. Resilience to the balance attack

The balance attack is a liveness attack exploiting Gasper's fork-choice rule formulated in [5]. This attack consists of balancing the votes of the validators on two parallel chains with the aim of avoiding the justification and finalization of the checkpoints. This balancing operation jeopardizes justifications as long as two-thirds of validators do not vote for the checkpoints. Therefore the aim of attackers is to keep honest validators split between two chains indefinitely, so that neither chain ever gets two-thirds of votes.

In our simulation with 13 nodes, we assume that 4 of them are dishonest (hence, less than one-third). The attack is triggered by a dishonest validator that proposes two blocks A and B for slot 0 of an epoch e and sending A to half of the nodes and B to the other half.

During e, honest validators will vote for their main view while dishonest validators do not send votes and collect the blocks of chain A and B into two separate subsets. In the $e+1$ epoch, the attacking validators parse these two sets and vote in order to balance the total votes of the two chains (see Fig. 10). By continuing to release votes with this delay, it will not be possible to obtain the two-thirds of votes on checkpoints and to select a canonical chain. The same results have been found in [5], where the authors simulate the attack in a synchronous network with 100 nodes and show that finalizations are not possible.

Ethereum has already mitigated the balance attack by introducing the so-called *proposer boost* [23]. The idea is the following. When a block is created, it gets a relevant increase of weight if it is received in time (in its own slot). Since the weight is used by the LMD-Ghost algorithm to solve forks, the heavier block will have more chance to be selected by nodes. In Fig. 11 we apply a boost to the block A of 40% of the stake of voters at epoch 0.

In our case, having all validators with stake equal to 32 ETH and having 3 validators with voting rights in each slot, the boost value is set to 38. The blue line highlights that, when the proposer boost starts, the honest nodes begin to vote for the chain A. The same trend is pictured in Fig. 12, where the probability of finalizing a checkpoint is 1 after 3 epochs.

5.5. Resilience to the bouncing attack

The bouncing attack [4] is another liveness attack that can significantly disrupt the normal operation of a network. In this case, the aim of the attackers is to bounce from one chain to another, not allowing the protocol to finalize checkpoints.

We simulate the attack with 13 nodes, of which 4 are dishonest, and we assume epoch $e = 0$ as the starting epoch. In this epoch, 2 checkpoints are created. In epoch $e+1$ the attackers wait until all honest validators have sent their checkpoint votes before releasing theirs, in order to justify one of the two checkpoints. Let us assume checkpoint A has two-thirds of votes. Meanwhile, during this epoch $(e+1)$, the validators continued to propose blocks on the two parallel chains, again obtaining two different checkpoints. Therefore, by repeating the previous steps, the dishonest validators will release their checkpoint votes again with delay, this time justifying the checkpoint on the other chain, the chain B. In this way, avoiding justifying two consecutive blocks of the same chain by "bouncing" from the chain A to B, it is

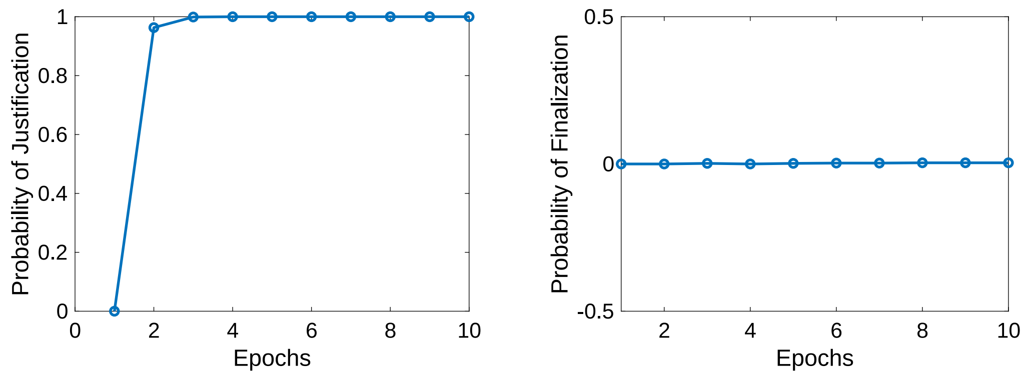

Fig. 14. Probability of justifying (left) and finalizing (right) a checkpoint during a bouncing attack.

possible to affect the liveness of the network. Fig. 13 shows the bounce in checkpoint justifications, going from chain A to chain B.

Fig. 14 (left) reports the probability of justifying a checkpoint and Fig. 14 (right) displays the probability of finalization. We derive that the ability to finalize is definitely compromised, thus confirming the results of [4]. This attack has been fixed in Ethereum by introducing a maximum slot limit in which each validator can send its checkpoint vote [4]. However the patch has been removed because the bouncing attack can be hardly reproduced in Gasper.

5.6. Resilience to the time attack

We finally study the resilience of Gasper to a *time attack*. Time-based attacks are designed to exploit the lack of a specific clock synchronization system in Ethereum to hamper the reachability of a finalization state. In particular, we use two well known attacks to the *Network Time Protocol*, which is one of the protocol used by Ethereum validators to synchronize clocks [24,25]. The goal of these attacks is to de-synchronize clocks of validators in order to slow down them for two epochs, thus becoming inactive. Assuming to have 13 validators, the steps of our attack are the following:

1. *Picking dishonest validators*. Seven validators are chosen to behave dishonestly, that is they aim to finalize a chain of block that is different than the actual one. Technically they behave honestly, since they just vote for different checkpoints. We notice that these validators cannot control the protocol because they do not own the two-thirds of the total stake.
2. *Slowing down honest validators*. The behavior of the other six honest validators is delayed by 64 slots. This means that the operations of block creation, voting for checkpoints, and retrieval of blocks are delayed of 64 slots, which is greater than the inclusion delay (32 slots). Therefore the validator appears inactive to the network. Technically, this attack is implemented by updating the rates of attacked validators (*e.g.* in Listing 2 the rate `rC_i` moves from 1/32 to 1/64 and the probability 1 in line 7 becomes 1/32).
3. *Inactivity leak*. The attacked validators lose their stake because of the inactivity. When the stake of these validators is less than 16 ETH, they transit to *Exit* (this is a new ad-hoc state specifically created for this attack) which precludes them from future operations within the network and removes their stake from the total stake. To speed-up this step, we decided to penalize inactive validators by 1 ETH (a considerably higher penalty than that for inactivity, but necessary to simulate this phenomenon in less time). This step is performed until the stake of the dishonest validators is less than two-thirds of the total stake.
4. *Full control of the network*. Once two-thirds of the total stake is retained by the dishonest validators, the attack is complete.

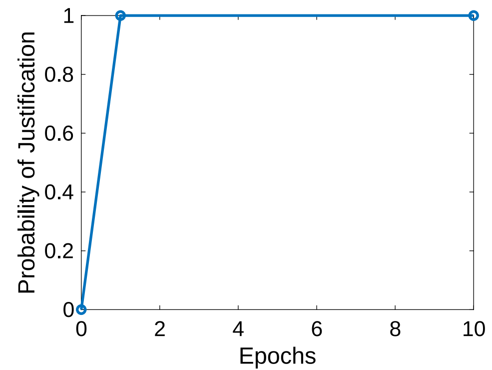

Fig. 15. Probability of justification for honest validators before a time attack.

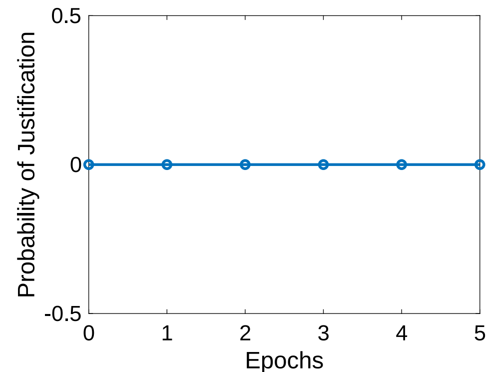

Fig. 16. Probability of justification for honest validators during a time attack.

Fig. 15 highlights the number of justified blocks before the attack starts. In this case the probability of justification is 1 (at epoch 1).

However, as soon as validator's clocks are slowed or affected by the time attack, they can no longer participate in the justification process and Fig. 16 shows that the justification probability becomes 0. Since, in Gasper, the operations must occur within fixed time intervals, the Time Attack rises a substantial concern about the effectiveness of the validator.

Fig. 17 displays another phenomenon: the number of blocks justified by dishonest validators becomes larger and larger over time as inactivity penalties are assigned delayed validators.

Indeed, this mechanism gives penalties to validators with notably slower operational clocks within the network, categorizing them as *inactive*. In turn, this categorization ultimately paves the way for malicious validators to secure their checkpoint. Of note is that this inactivity process, resulting in forced exit of inactive validators, takes three weeks

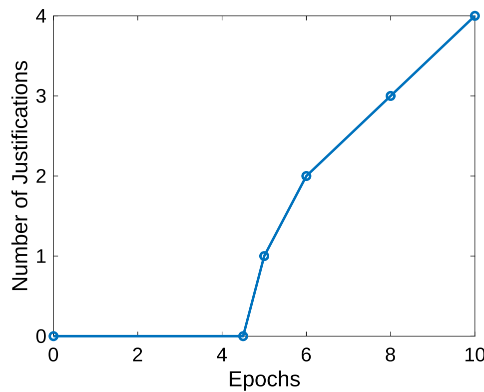

Fig. 17. Number of justifications made by the dishonest validators.

to complete [26]. In our simulation, the process was speeded up by increasing penalties for inactive validators.

6. Related works

The analysis technique used for Gasper is similar to those applied to Bitcoin and Hybrid Casper in [6,7].

A similar technique has been used to analyze Tendermint [27], where the authors verify that the consensus protocol is deadlock-free and is able to reach consensus when at least two-thirds of the network nodes are in agreement. They also prove that a minority set of nodes, which is greater that one-third, is enough to censor the majority of the network and prevent the network from reaching consensus. As in our modeling, the network of nodes is a parallel composition of processes; however, in [27], the system is verified by means the PAT model checker [28], which is not probabilistic.

A first machine checked proof for safety and liveness properties of a consensus algorithm has been delivered in [29]. They verify a Proof of Stake protocol, using the Coq proof assistant [30]. In particular, they consider a synchronous network with a static set of corrupted parties, define an operational semantics for this setting and prove *chain growth*, *chain quality*, and *common prefix*. The chain growth property states that the length of the main 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 have been created by honest nodes. 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). With respect to our work, the setting in [29] is synchronous, which assumes that messages are delivered within a known fixed time bound, and probabilities are not taken into account, meaning that there is no modeling or analysis of stochastic aspects of message delivery and parties' behavior. In contrast, our approach accounts for probabilistic scenarios and allows for an asynchronous setting where there is no fixed upper bound on message delivery times.

Coq has also been used to verify the Algorand PoS protocol [31]. In particular, Algorand correctness is the so-called *asynchronous safety*, *e.g.* two different blocks can never be certified in the same round (certification is a notion analogous to finalization in Gasper). In [31], asynchronous safety is demonstrated by taking into account network delays, timing issues, and malicious nodes. However, in the case of Algorand proofs, different settings require completely new demonstrations. In contrast, 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. This adaptability is a key strength of our approach, providing a more flexible and comprehensive verification framework.

Security guarantees have been also thoroughly studied for the PoS protocol Ouroboros [32]. In the case of Ouroboros, the studied properties are *persistence*, meaning that, once an honest node declares a given transaction as "stable", then all the other honest nodes will agree on that choice; and *liveness*, meaning that, once an honestly generated transaction has been made available to the network, then it will become eventually stable. The proofs that Ouroboros enjoys these properties are manual. We think that our `PRISM+` model checker can be used to study the foregoing properties in an automatic way. Up-to our understanding, it will be sufficient to change the process of choosing the next node that can propose a block.

A technique similar to our one but using the PEPA model checker has been applied to examine the incentives that drive miners to invest resources in Proof-of-Work (PoW) blockchains to maximize profits [33]. Notably, [33] studies the incentive mechanisms without the modeling of the blockchain and its blocks. In contrast, we take a different approach. We dive into the internal dynamics of the blockchain, offering a comprehensive analysis of its functioning and a broader spectrum of analyses. Additionally, our methodology is versatile and has also been successfully applied to analyze PoW systems and PoS ones.

7. Conclusion

We verified the Gasper protocol using the stochastic model checker `PRISM+`, which includes specialized primitives for representing the data types `ledger` and `block`, along with their associated operations.

Our experimental simulations examined the impact of network latency on finalizations, providing empirical evidence of Gasper's robustness. We tested the protocol under various time delays and verified that the probability of justification and finalization significantly decreases as the time required for a block to be delivered increases. Furthermore, we analyzed stake updates and discovered that while larger networks yield higher rewards for proposing a block, smaller networks increase the likelihood of becoming a proposer and receiving substantial rewards. We evaluated the protocol's resilience against balance, bouncing, and time attacks, showing that the protocol is highly susceptible to these attacks. These findings validate results from the literature [4,5], and we further analyzed the effectiveness of the proposed patches.

Our analysis is not exhaustive. Future work will focus on exploring the protocol's behavior in dynamic networks, where nodes may enter or exit the network. Also assessing Gasper's resilience against emerging threats and adversarial strategies that may arise in more complex and evolving network environments is relevant. Another research direction that we recently found interesting is the question of fairness of the Gasper protocol. In this protocol, the ability to generate blocks is proportional to the wealth on stakes of a validator. Therefore a validator with higher stake has more chance to be selected as a proposer and to increase his own stake. As a consequence, the distance in wealthiness augments, thus reducing the ability of participants with smaller stakes to create blocks (the fairness decreases). By using the `PRISM+` model of Gasper we actually verified that the wealthiest validators consistently maintain their advantageous positions while the least affluent validators struggle to improve their standings [34]. This emphasizes the need to design, implement and analyze mechanisms for greater equity and inclusivity in blockchain ecosystems.

A last research direction is about the correctness of PoS protocols other than Gasper. According to the Ethereum Foundation, Gasper consumes about the 99.95% less energy than the current PoW protocols. For this reason PoS protocols have been already developed for pervasive computing systems where energy consumption is an issue [35,36]. However, while extensive simulations have been undertaken, there is no formal analysis about the robustness of such proposals when basic parameters change. We intend to pursue this goal by means of `PRISM+`, following the technique used for Gasper.

CRediT authorship contribution statement

Cosimo Laneve: Writing – review & editing, Writing – original draft, Supervision, Methodology, Funding acquisition, Formal analysis, Conceptualization. **Adele Veschetti:** Writing – review & editing, Writing – original draft, Validation, Software, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Cosimo Laneve reports financial support was provided by European Union NextGenerationEU. Adele Veschetti reports financial support was provided by ATHENE. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, 2008, https://bitcoin.org/bitcoin.pdf.
[2] C.C. for Alternative Finance, CambridgeBitcoin electricity consumption index, 2021, (last access 2021). https://cbeci.org/.
[3] J. Krumm, Ubiquitous Computing Fundamentals, CRC Press, 2018.
[4] U. Pavloff, Y. Amoussou-Guenou, S. Tucci Piergiovanni, Ethereum proof-of-stake under scrutiny, in: SAC 2023, ACM, 2023, pp. 212–221.
[5] J. Neu, E.N. Tas, D. Tse, Ebb-and-flow protocols: A resolution of the availability-finality dilemma, 2020, CoRR abs/2009.04987.
[6] S. Bistarelli, R.D. Nicola, L. Galletta, C. Laneve, I. Mercanti, A. Veschetti, Stochastic modeling and analysis of the Bitcoin protocol in the presence of block communication delays, Concurr. Comput. Pr. Exp. 35 (16) (2023).
[7] L. Galletta, C. Laneve, I. Mercanti, A. Veschetti, Resilience of Hybrid Casper under varying values of parameters, Distrib. Ledger Technol. Res. Pr. 2 (1) (2023) 5:1–5:25.
[8] M.Z. Kwiatkowska, G. Norman, D. Parker, Probabilistic symbolic model checking with PRISM: A hybrid approach, in: TACAS, in: LNCS, vol. 2280, Springer, 2002, pp. 52–66.
[9] C. Laneve, S. Solmonte, A. Veschetti, A stochastic analysis of the gasper protocol, in: IEEE PerCOm – 5th Workshop on Blockchain TheoRY and ApplicatIONS, BRAIN 2024, IEEE, 2024, pp. 518–523.
[10] A. Zhang, RANDAO: A DAO working as RNG of ethereum, 2022, https://github.com/randao/randao.
[11] Ethereum Documentation, *Proof-of-stake rewards and penalties*, 2024, https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/rewards-and-penalties/.
[12] M.Z. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: Verification of probabilistic real-time systems, in: CAV, in: LNCS, vol. 6806, Springer, 2011, pp. 585–591.
[13] C. Decker, R. Wattenhofer, Information propagation in the Bitcoin network, in: P2P, IEEE, 2013, pp. 1–10.
[14] A. Aziz, K. Sanwal, V. Singhal, R.K. Brayton, Verifying continuous time Markov chains, in: CAV, in: LNCS, vol. 1102, Springer, 1996, pp. 269–276.
[15] C. Baier, B.R. Haverkort, H. Hermanns, J. Katoen, Model checking continuous-time Markov chains by transient analysis, in: CAV, in: LNCS, vol. 1855, Springer, 2000, pp. 358–372.
[16] C. Baier, B.R. Haverkort, H. Hermanns, J. Katoen, Model-checking algorithms for continuous-time Markov chains, IEEE Trans. Softw. Eng. 29 (6) (2003) 524–541.
[17] C. Laneve, S. Solmonte, A. Veschetti, PRISM+ software package, supporting material, and additional experiments, 2023, URL https://github.com/adeleveschetti/ethereum-analysis.
[18] Ethereum Documentation, Rewards and penalties, 2023, https://ethereum.org/de/developers/docs/consensus-mechanisms/pos/rewards-and-penalties/.
[19] B. Edgington, A technical handbook on Ethereum's move to Proof of Stake and beyond, 2023, https://eth2book.info/capella/.
[20] Ethereum Documentation, Attestations, 2023, https://ethereum.org/se/developers/docs/consensus-mechanisms/pos/attestations/.
[21] V. Buterin, D. Reijsbergen, S. Leonardos, G. Piliouras, Incentives in Ethereum's Hybrid Casper protocol, Int. J. Netw. Manage. 30 (5) (2020).
[22] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, Y.X. Zhang, Combining GHOST and casper, 2020, CoRR, abs/2003.03052.
[23] V. Buterin, Proposal for mitigation against balancing attacks to LMD GHOST, 2020, https://notes.ethereum.org/@vbuterin/lmd_ghost_mitigation.
[24] J. Selvi, Bypassing HTTP strict transport security, Black Hat Eur. 54 (2014).
[25] A. Malhotra, I.E. Cohen, E. Brakke, S. Goldberg, Attacking the network time protocol, in: 23rd Annual Network and Distributed System Security Symposium, The Internet Society, 2016, pp. 1–19.
[26] P. Mccorry, The inactivity leak, 2023, https://www.cryptofrens.info/p/the-inactivity-leak.
[27] W.Y.M.M. Thin, N. Dong, G. Bai, J.S. Dong, Formal Analysis of a Proof-of-Stake Blockchain, in: ICECCS, IEEE Computer Society, 2018, pp. 197–200.
[28] J. Sun, Y. Liu, J.S. Dong, J. Pang, PAT: towards flexible verification under fairness, in: Computer Aided Verification, in: Lectures Notes in Computer Science, vol. 5643, Springer, 2009, pp. 709–714.
[29] S.E. Thomsen, B. Spitters, Formalizing nakamoto-style proof of stake, in: 34th IEEE Computer Security Foundations Symposium, CSF 2021, IEEE, 2021, pp. 1–15.
[30] T.C.D. Team, The coq proof assistant, version 8.19.2, 2024, URL https://coq.inria.fr.
[31] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, N. Zeldovich, Algorand: Scaling Byzantine agreements for cryptocurrencies, in: SOSP, ACM, 2017, pp. 51–68.
[32] A. Kiayias, A. Russell, B. David, R. Oliynykov, Ouroboros: A provably secure Proof-of-Stake blockchain protocol, in: CRYPTO (1), in: LNCS, vol. 10401, Springer, 2017, pp. 357–388.
[33] D. Smuseva, I. Malakhov, A. Marin, A. van Moorsel, S. Rossi, Verifier's dilemma in ethereum blockchain: A quantitative analysis, in: QEST, in: Lecture Notes in Computer Science, vol. 13479, Springer, 2022, pp. 317–336.
[34] S. Bistarelli, C. Laneve, I. Mercanti, A. Veschetti, Analyzing the fairness of proof of stake ethereum, in: Proc. of 6th Distributed Ledger Technology Workshop, Turin, 2024.
[35] Y. Huang, Y. Zeng, F. Ye, Y. Yang, Incentive assignment in hybrid consensus blockchain systems in pervasive edge environments, IEEE Trans. Comput. 71 (9) (2022) 2102–2115, http://dx.doi.org/10.1109/TC.2021.3122891.
[36] Y. Huang, J. Zhang, J. Duan, B. Xiao, F. Ye, Y. Yang, Resource allocation and consensus of blockchains in pervasive edge computing environments, IEEE Trans. Mob. Comput. 21 (9) (2022) 3298–3311, http://dx.doi.org/10.1109/TMC.2021.3053230.