A Stochastic Analysis of the Gasper Protocol

***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 modelling 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.**

***Index Terms*—stochastic modeling and analysis, proof of stake, blockchain fork**

I. 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 a consensus protocol that imposes a total order on the updates, which are called *blocks*. The network recently underwent a significant protocol transformation, transitioning from a Proof of Work (PoW) 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 $2/3$ 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 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 [1], [2]. 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 [3], [4], the automatic verifier we use is `PRISM+`, an extension of the stochastic model checker `PRISM` [5] 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, 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 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 [1], [2], and analyze the effectiveness of patches that have been proposed.

The paper is organized as follows. Section II contains an overview of Gasper. The `PRISM+` model of Gasper is defined in Section III and the coherence of the model and the analyses on the security of the protocol are reported in Section IV. Section V compares our proposal with the literature and Section VI draws some conclusions and discusses possible future work.

II. 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 irreversible storage of blocks in the blockchain, called finalization, and the voting process of blocks;
C. the incentive mechanism that rewards honest validators and punishes misbehaving ones.

We overview these steps in some detail in the following subsections. For lack of space, aspects related to dynamics of the network (validators that enter or exit from the protocol) are overlooked.

A. 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 seconds, 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 seconds 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 hight 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 [6]. The 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 $e+3$.

B. Finalization mechanism and the voting process

Finality refers to a critical property of checkpoints that renders 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. This vote is used to solve forks in the ledger. In particular, for each block vote, a weight is added to each block of the chain that has the block voted as a descendent. (The weight is proportional to the stake of the corresponding validator.) Then, in case of forks, the so-called LMD-Ghost algorithm identifies the main chain by choosing the one with higher weight.

C. Incentives and penalties

Validators earn rewards through various activities, including making consistent votes with the majority of other validators in correspondence of checkpoints and proposing blocks. The reward values for these activities within each epoch are computed according to the so-called $\texttt{base\_reward}_i$. This fundamental unit represents the average reward that a validator would receive under optimal conditions per epoch and it is proportional to the effective balance of the validator and inversely proportional to the number of validators on the network:

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

Moreover, if a validator votes a different block from the majority of the nodes, it will receive a penalty, which is the same amount of the corresponding reward: [1]

$$\begin{aligned} \texttt{reward}_i = \; & (3 \times \texttt{base\_rew}_i \times \texttt{stake}_i) / \textstyle\sum_{j \in \texttt{V}} \texttt{stake}_j \\ & + 7/8 \times \texttt{base\_rew}_i \\ \texttt{pen}_i = \; & -3 \times \texttt{base\_rew}_i \end{aligned}$$

The reward a proposer receives when one of the blocks it created has been validated can be computed by the following formula:

$$\texttt{reward\_bp}_i = \texttt{base\_rew}_i + 1/8 \times \sum_{j \in \texttt{V}} \texttt{reward}_j$$

Thus, at every epoch, the stake of a validator i is either $\texttt{stake}_i + \texttt{reward}_i$, if it has been a honest validator, or $\texttt{stake}_i + \texttt{pen}_i$, if it has been dishonest, or $\texttt{stake}_i + \texttt{reward\_bp}_i$ if it has been a proposer whose block has been validated. The computations of rewards and penalties are taken from the official Ethereum documentation [7], [8] (that have been also verified in real-world scenarios in [9]).

Gasper also features a more drastic penalty scenario, called *slashing*, which causes the expulsion of a validator from the network, accompanied by the retrieval of its stake. 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 stake is immediately burned and it is banned for 36 days; during this period additional penalties are also applied.

For example, one such penalty is get when a validator doesn't 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.

III. The Gasper model in PRISM+

Gasper is modelled in `PRISM+` 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 our system, contributing to the overall robustness and reliability of our implementation. The architecture of our model is in Figure 1. The module

[1] In Gasper there are rewards and penalties within committees as well, which use aggregators to get a consensus between components. We do not discuss this issue because we are assuming singleton components.

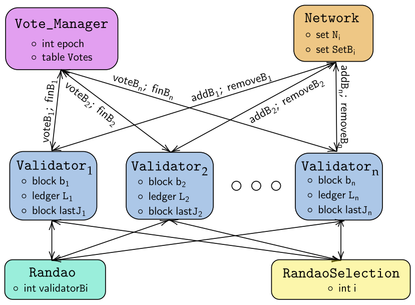

Fig. 1: The Ethereum PoS model architecture.

`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 IV; $\texttt{Validator}_\texttt{i}$ are the processes modeling the validators and `Randao` and `RandaoSelection` simulate the random selection of the proposers for the next epoch.

In our model, blocks are triples $(\texttt{v}^\texttt{n};\texttt{p};\texttt{h})$, 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 $(\texttt{v}^\texttt{n};\texttt{p};\texttt{h})$ points to; `h` is the *height* of the block in the ledger. In turn, ledger are tuples $\langle\texttt{T};\texttt{f};\texttt{p}\rangle$ where `T` is a tree of blocks; `f` is the name of a block in `T` (the *last finalized block* of `L`) and `p` is the *handle* of `L`, *i.e.* the name of a leaf block at maximal height in the subtree rooted at `f`.

For clarity sake, we present a sugared version of the PRISM+ code of Gasper; the online repository [10] contains the actual implementation, the verified properties with the data of our analyses and the instructions for the use of the tool.

A. *The* `Validator`

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

Listing 1: The code of a `Validator`.

The $\texttt{Validator}_\texttt{i}$ module is reported in Listing 1. A validator 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 transits to the `Create` state (line 10), or (*ii*) has been selected to vote in this slot and transits to `Vote` (line 11), or (*iii*) receive a new block from the network and transits to `Receive` (line 12), or (*iv*) check whether a checkpoint can be justified and transit to `Check` (line 13), provided the block is the end of an epoch. The rates **`rC_i`** represent how often a validator should check for new justified/finalized blocks. In the Gasper protocol [11], in the absence of excessive delays or congestion, this happens within the slot to which they have been assigned. Thus $\texttt{rC\_i} = \texttt{s}/(len_{epoch})$, 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 14-15) 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 1/12.6, which is the broadcast delay of the Bitcoin network [12], in order to have an average delay value that allows us to compare our simulation with other models. However, in Section IV, we will consider different latencies to test the protocol under different settings. If the new block is a checkpoint (the height of the block is a multiple of $len_{epoch}/\texttt{s}$, 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 16).

In the state `Receive`, the validator 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 `Move`, `Validator` verifies whether the block can be inserted in its own ledger (lines from 20-21). If this is the case, `Validator` adds the block and returns to the initial state (line 22). 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 $\texttt{C}_L$, has received the majority of the votes (i.e. $\texttt{C}_L$ has been justified) and whether the last but one checkpoint in the blockchain of $\texttt{C}_L$, say $\texttt{C}_A$, is also justified. If this is the case, $\texttt{C}_A$ becomes finalized and $\texttt{C}_L$ becomes justified – in lines 23-24 this is performed by storing $\texttt{C}_L$ 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.

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

Listing 2: The code of Randao.

```
module Randao_Selection

    i : [0..32] init 0;

    [] (i=0)&(randao=false) -> 1 : (validatorID'=ValidatorB0)&(i'=i+1);
    for j from 1 to 31:
        [] (i=j) -> slot :
            (validatorID'=ValidatorB_j)&(voterID_0'=Voter_0i)
            &(voterID_1'=Voter_1i)&(voterID_2'=Voter_2i)&(i'=i+1);
        [] (i>31) -> 1: (i'=0)&(randao'=true);
endmodule
```

Listing 3: The code of RandaoSelection.

If C_A is not justified then C_L is stored in lastJ_i (lines 25-26). In any case, the validator returns to Start and the process starts again.

B. The RANDAO

The RANDAO process has been split in two different modules: Randao and RandaoSelection. The first one, reported in Listing 2, 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 12–18 in Listing 2). 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 next block proposer.

The code in Listing 3 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.

C. The Vote_Manager

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

```
module Vote_Manager
   Stakes : map {} ;
   Votes : map {};
   epoch = 0 ;
   for i from 0 to N:
      Stakes[validator_i] : [0..MAX_STAKE] init STAKE_i;
   for i from 0 to N:
      [voteB_i]  -> 1 : Votes'=addVote(Votes,b_i,Validator_i);
      [finB_i] (height(lastF(L_i)) > epoch) ->
         1: epoch'=height(lastF(L_i))&Stakes'=updateS(Stakes,Votes,lastF(L_i));
      [finB_i] (height(lastF(L_i)) <= epoch) -> 1:  ;
endmodule
```

Listing 4: The code of Vote_Manager.

The transitions involved in this module are repeated for each validator, as can be imagined from the pseudocode. The transition voteBlock, when the Updater is ready to work, all it does is add a vote inside the hash map for that specific block passed as input.
Much more complex is the finBlock transition which, after a careful analysis of the consistency regarding the height of that block in relation to the other checkpoints or finalized blocks, allows you to update the stake value of the single validator and consequently also that of the variable tot_stakes.

D. The Network

The Network module manages the entire part relating to the addition or elimination of a block to the main blockchain, considering the addition or removal delays that are foreseen by the protocol used.

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

Listing 5: The code of Network.

The transitions in this module are used to manage the step before justifying or finalizing a block. If a block needs to be added, in the addBlock transition, then that block will be added to each validator's set, so that everyone can have it in their set. The opposite situation exists in extractBlock, which is carried out only in the set of the user who proposed the block, i.e. the validator with the role of proposer, given that it is an operation that is called chronologically before the addition.

E. 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 we are interested in. In particular, since our goal is to study the behaviour 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;
2) the network consists of validators that also create new blocks (in particular, we drop the distinction between

validators and proposers); (we also drop 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 II-A).

Additionally, we notice that

- the slashing protocol of Section II-C has not been reported. Actually it is implemented in the repository [10] (by refining both $\texttt{Validator}_\texttt{i}$ and `Vote_Manager`) and we use the refined algorithm to study the time attack of Section IV-F;
- we have not analyzed dynamic networks, that is validators that may enter or exit the protocol (*c.f.* end of Section II). Therefore, there is no module corresponding to `Deposit`. The modelling and the corresponding analysis is left to future work.

IV. 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 are conducted with a limited number of validators (13), which is the same number used in Hybrid Casper tests [4]. However, we notice that when the number of validators increases (*e.g.* 16), the overall trend does not change (in case of 16 validators, the differences are in the order of 10^{-3}).

A. Coherence of the model – comparison with Hybrid Casper

Hybrid Casper [13] 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 seconds. 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 time-frames 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 14s because the PoW may be longer). Therefore, we expect that Gasper has a better performance in proposing and finalizing blocks. The following analyses will confirm this expectation.

The results obtained by analyzing Gasper (blue lines) are compared with dose in the literature [14] whenever available (yellow line) and with those obtained for Hybrid Casper, reported in [4] (red line).

As regards *block creation*, Figure 2 reports the process over a duration of 25s (a bit more of two Gasper slots and a bit less of two Hybrid Casper slots).

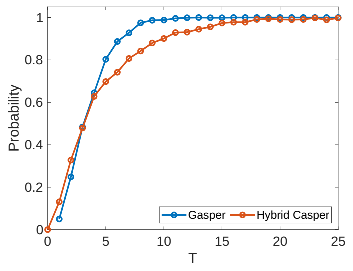

Fig. 2: Block creation probability over time.

We notice that a block is created in Gasper within 12s with probability of 1. On the contrary, in Hybrid Casper the probability of creating a block within 14s 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 14s.

As regards *forks*, Figure 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

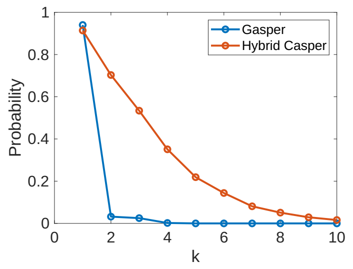

Fig. 3: Probability of a fork of length k.

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 seconds. In Figures 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

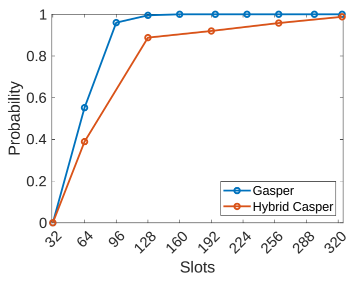

Fig. 4: Justification.

accordance with [13] where this probability is stated to be greater than 0.5 in one epoch.

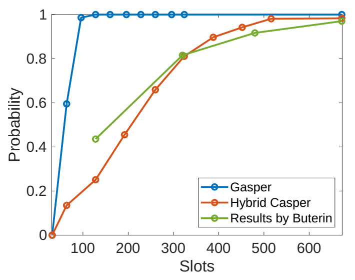

Fig. 5: Finalization.

In Figure 5, we compare the probability of finalizing blocks also with the results presented in [13]. In our Gasper model, the probability of finalizing a checkpoint is already greater than 0.6 after 32 slots, which is due to the process of creation blocks: 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 Figure 3) and, thus, a higher probability of finalizing checkpoints.

B. Different Network Latency

In order to analyze how the system is affected by the network latency, we report some analyses considering 4 different settings: no delay, a delay of 12.6 seconds, a delay of 25.2 seconds and a delay of 37.8 seconds. The process of creating blocks is not affected by the delay as is probability of having a state of fork, due to the fact that the blocks are created by using the RANDAO smart contract.

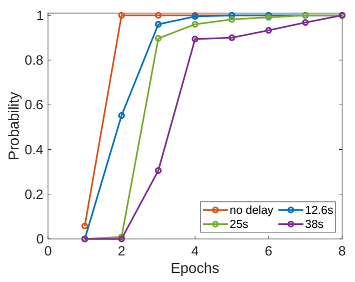

Fig. 6: Justification with different delays.

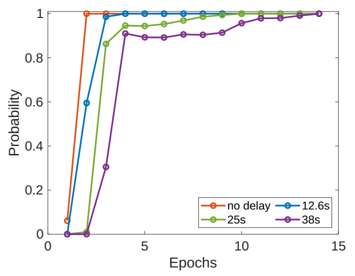

Fig. 7: Finalization with different delays.

Figures 6 and 7 report 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 seconds 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.

C. 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 at the peak of individual epochs. In the experiments of Figures 8– 11, we consider a network of honest validators with probability 0.8 of sending correct attestations and 0.2 of sending incorrect ones. 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 based on whether all the votes by the validators are correct. The computations of rewards and penalties that we use for the following analyses are presented in Section II.

The graphical representations of the following figures will show the value $\texttt{stake}_i$ of the validator i. On the x axis, we will indicate the time intervals in epochs, with the understanding that one slot equals 12 seconds and one epoch equals 32 slots (hence 384 seconds). Figure 8 spots the average stake of a

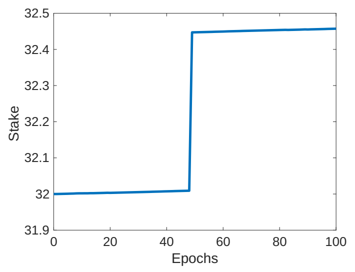

Fig. 8: $\texttt{stake}_i$ with 100k validators having equal initial stake.

validator in 100 epochs, assuming there are 10^5 validators with a stake of 32 ETH. In this simulation, the base reward is 9050

Gwei (1 Gwei is equal to 10^{-9} ETH), the reward of a correct attestation of a block is 30600 Gwei, while the proposer of the block is rewarded 0.375 ETH. In case of incorrect attestation, the penalty is 22010 Gwei, which are removed from the stake of the corresponding validator. We observe that, in Figure 8, 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. Figure 9 reports the average stake of a validator in 100 epochs, assuming there are 0.5×10^5 validators with a stake of 32 ETH In this simulation,

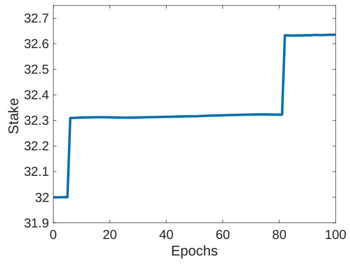

Fig. 9: $\texttt{stake}_i$ with 50k validators having equal initial stake.

the base reward is 12800 Gwei, so 49600 in the case of a correct attestation and 0.310 ETH with the proposal of a block. In case of incorrect attestation, the penalty value is equivalent to 38400 Gwei.

The slopes in Figure 9 indicate that the validator has became a block proposer twice in 100 epochs, around epochs 10 and 80, therefore exceeding the 32.5 ETH reached in the experiment with 100 thousand 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.

Figure 10 shows the performance of a single validator with a stake higher than that of the others, in this case 50 ETH. The trend is always analyzed over 100 epochs, considering the same reward for the block proposal as before, since it is mainly based on the attestations' rewards of the validators that attest to that block [8]. The base reward of the richest validator is 20000 Gwei, while for a valid attestation, it is 77500 Gwei. The penalty is set at 38400 Gwei. This experiment highlights that the presence of a unique validator with a larger initial stake does not let it to enrich considerably more than the others, given that the stake amount influencing the proposer's selection among validators is 32 ETH.

The last experiment of this stake analysis, shown in Figure 11, shows the trend of the average stake considering that all validators are richer, considering a per capita stake of 64ETH. In this experiment, the base reward is 18102 Gwei, and therefore 70145 Gwei for valid attestations. The penalty for incorrect attestations is 53306 Gwei. Considering that all

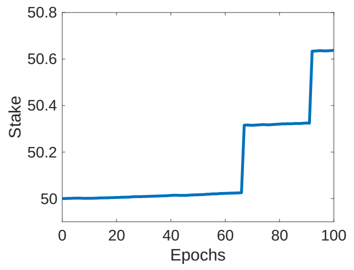

Fig. 10: $\texttt{stake}_i$ with 50k validators and initial value of 50 ETH.

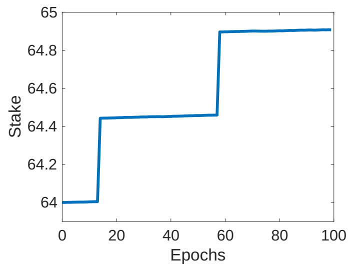

Fig. 11: $\texttt{stake}_i$ with 50k validators, all with stake 64 ETH.

validators have a higher stake, the reward for proposing blocks increases to 0.4384 ETH. In 100 epochs and with two block proposals by the single validator shown, the threshold of 1 ETH earned in total is almost reached. From this experiment, we can deduce that as the average total stake value increases, in this case, double that expected to become a validator and enter the network, the gain for a single validator grows significantly, leading to a large injection of ETH and consequently to all the results that this operation will achieve.

D. Resilience to the Balance Attack

The balance attack is a liveness attack exploiting Gasper's fork-choice rule formulated in [2]. 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 $\frac{2}{3}$ 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 ($< \frac{1}{3}$). 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, thus starting a fork. 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 Figure 12). By continuing to release votes with this delay, it will not be possible to obtain the $\frac{2}{3}$ of votes on checkpoints and to select a canonical chain.

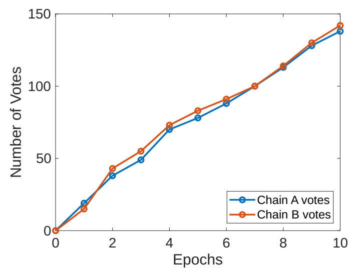

Fig. 12: Balance between votes during a balance attack.

The same results have been found in [2], where the authors simulate the attack in a synchronous network with 100 nodes and show that finalizations are not possible.

The balance attack has been mitigated in Ethereum by introducing the so-called *proposer boost*. 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 Figure 13 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.

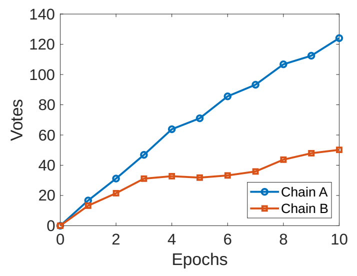

Fig. 13: Trend of the votes for the two chains A and B in 10 epochs.

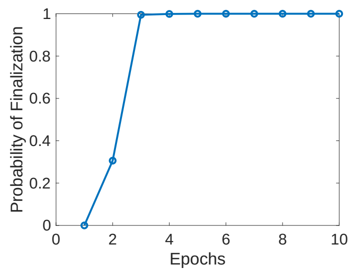

Fig. 14: The difference in votes received between the two chains allows liveness to be re-established

The same trend is pictured in Figure 14, where the probability of finalizing a checkpoint is 1 after 3 epochs.

E. Resilience to the Bouncing Attack

The bouncing attack [1] 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 ($< \frac{1}{3}$), 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's assume checkpoint A has $\frac{2}{3}$ 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 possible to affect the liveness of the network. Figure 15 shows the bounce in checkpoint justifications, going from chain A to chain B.

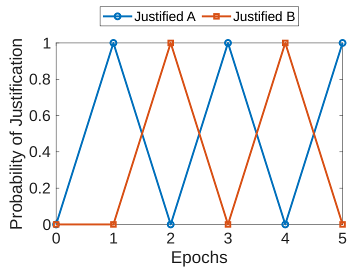

Fig. 15: Phenomenon of bouncing from chain A to chain B.

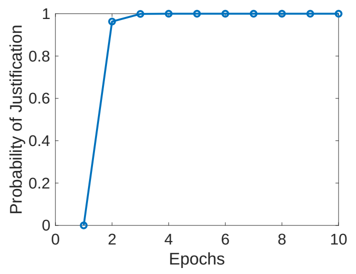

Fig. 16: Probability of justify a checkpoint.

Figure 16 shows the probability of justifying a checkpoint and Figure 17 displays the probability of finalizing a checkpoint. We can deduce that the ability to finalize is compromised, thus confirming the results of [1]. We notice that this attack has been fixed by introducing a maximum slot limit in which each validator can send its checkpoint vote. However the patch was later removed because the bouncing attack can be hardly reproduced in Gasper.

F. Resilience to the Time Attack

In this section, we study the resilience of Gasper to a *time attack*. Time-based attacks are designed to exploit the lack

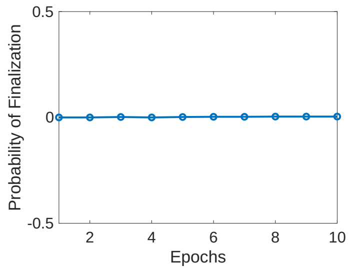

Fig. 17: Probability of Finalize a checkpoint.

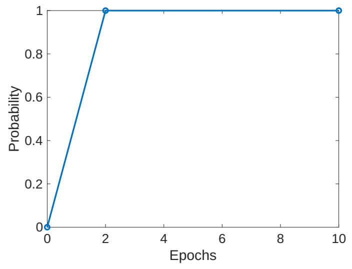

Fig. 18: Honest Validators' Justifications in Time Attack.

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 [15], [16]. According to these attacks, clocks of validators may be de-synchronized, therefore a validator may be slow down 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: they just vote for a different checkpoints. We notice that these validators cannot control the protocol because they do not own the 2/3 of the total stake.
2) *Slowing down honest validators*. The behaviour 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 reater 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 1 the rate `rC_i` moves from 1/32 to 1/64 and the probability 1 in line 8 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 2/3 of the total stake.
4) *Full control of the network*. Once 2/3 of the total stake is retained by the dishonest validators, the attack is complete.

Figure 18 highlights the number of justified blocks by honest before the attack starts. In this case the probability of justification is 1. But as soon as their clocks are slowed or affected by this attack, they can no longer participate in the justification process. Figure 19 displays the probability of justification of an attacked honest validator. Since in Gasper, the operations must occur within fixed time intervals, the Time Attack rises a substantial concern about the effectiveness of the validator.

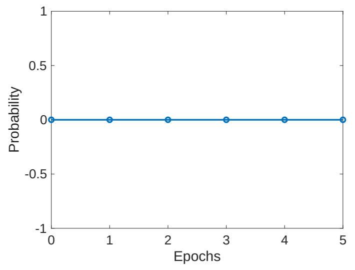

Fig. 19: Honest Validators' Justification rate during Time Attack.

In Figure 20, we see that the number of blocks justified by dishonest validators develops a larger increase over time as inactivity penalties are assigned. Initially, this mechanism imposes penalties on validators with notably slower operational clocks within the network, categorizing them as *inactive*. 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 to complete [17]. In this simulation, the process was speeded up by increasing penalties for inactive validators. Overall, we notice that, as the stakes of the validators increase, the rewards for the validators increases as well, leading to a large injection of ETH in the system. In particular, the rewards are larger when the increases of stakes concern the majority-of/all-the validators.

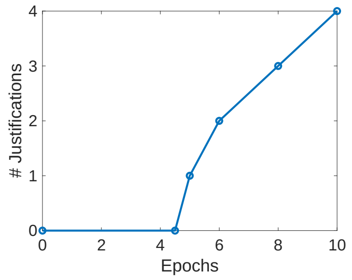

Fig. 20: Justifications during a Time Attack.

V. Related Works

For what concerns Gasper, to the best of our knowledge, there are few contributions that analyze its correctness using formal methods. We discuss them below.

The analysis technique that has been used for Gasper is similar to the one in [3], [4] for Bitcoin and Hybrid Casper, respectively. Also Tendermint, another PoS protocol, has been analyzed using a similar technique in [18]–[20]. In that case, the network of validators is modelled as a parallel composition of processes and the system is verified by means the PAT model checker [21], which is neither stochastic nor probabilistic. On the contrary, the Algorand PoS protocol [22] has been verified with the Coq theorem prover. Its correctness, *i.e.* two different blocks can never be certified in the same round even when the adversary completely controls the network (asynchronous safety), is also demonstrated taking into account network delays, timing issues, and malicious nodes. We think that our analysis can be easily adapted to Algorand once the protocol has been modelled in `PRISM+`. Finally, a PoS protocol with rigorous security guarantees is Ouroboros [23], where the authors prove some security properties through manual proofs. We are confident that we can adopt our methodology to study properties of Ouroboros as well.

VI. Conclusion

We have analyzed the Gasper, the recent consensus protocol adopted by Ethereum. The protocol has been verified by means of the stochastic model checker `PRISM+` that has ad-hoc primitives for expressing the data types `ledger` and `block` and the operations upon them.

The findings, derived from experimental simulations, confirmed the fidelity of this simulation to Gasper's specifications. We analyze how justification and finalization are affected by the network latency and test the protocol against the balance, bouncing and time attacks. One noteworthy observation that emerged from our investigation was the Gasper's ability to incentivize honest validators, fostering gradual stake growth over time. This growth of stakes follows linear trajectory rather than experiencing an exponential ascent, regardless of their ETH holdings.

We think that our results contribute to the ongoing discourse surrounding Ethereum's PoS, offering valuable insights into the practicality, security, and resilience embedded in the Gasper protocol.

It is essential to recognize that our analysis is preliminary and not exhaustive. We plan to explore the protocol in presence of nodes that may enter or exit the network (dynamic network) and assessing Gasper's resilience against emerging threats.

References

[1] U. Pavloff, Y. Amoussou-Guenou, and S. Tucci Piergiovanni, "Ethereum Proof-of-Stake under Scrutiny," in *SAC*. ACM, 2023, pp. 212–221.

[2] J. Neu, E. N. Tas, and D. Tse, "Ebb-and-Flow Protocols: A resolution of the availability-finality dilemma," *CoRR*, vol. abs/2009.04987, 2020.

[3] S. Bistarelli, R. D. Nicola, L. Galletta, C. Laneve, I. Mercanti, and A. Veschetti, "Stochastic modeling and analysis of the Bitcoin protocol in the presence of block communication delays," *Concurr. Comput. Pract. Exp.*, vol. 35, no. 16, 2023.

[4] L. Galletta, C. Laneve, I. Mercanti, and A. Veschetti, "Resilience of Hybrid Casper under varying values of parameters," *Distributed Ledger Technol. Res. Pract.*, vol. 2, no. 1, pp. 5:1–5:25, 2023.

[5] M. Z. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic symbolic model checking with PRISM: A hybrid approach," in *TACAS*, ser. LNCS, vol. 2280. Springer, 2002, pp. 52–66.

[6] A. Zhang, "RANDAO: A DAO working as RNG of Ethereum," https://github.com/randao/randao, 2022.

[7] Ethereum Documentation, "Rewards and Penalties," https://ethereum.org/de/developers/docs/consensus-mechanisms/pos/rewards-and-penalties/, 2023.

[8] B. Edgington, "A technical handbook on Ethereum's move to Proof of Stake and beyond," https://eth2book.info/capella/, 2023.

[9] BitMEX Research, "Ethereum's proof of stake system: Calculating penalties and rewards," https://blog.bitmex.com/ethereums-proof-of-stake-system-calculating-penalties-rewards/, 2021.

[10] A. Authors. (2023) `PRISM+` software package, supporting material, and additional experiments. [Online]. Available: https://anon-github.automl.cc/r/ethereum-analysis-E840/

[11] Ethereum Documentation, "Attestations," https://ethereum.org/se/developers/docs/consensus-mechanisms/pos/attestations/, 2023.

[12] C. Decker and R. Wattenhofer, "Information propagation in the Bitcoin network," in *P2P*. IEEE, 2013, pp. 1–10.

[13] V. Buterin, D. Reijsbergen, S. Leonardos, and G. Piliouras, "Incentives in Ethereum's Hybrid Casper protocol," *International Journal of Network Management*, vol. 30, no. 5, p. e2098, 2020.

[14] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang, "Combining GHOST and casper," *CoRR*, vol. abs/2003.03052, 2020.

[15] J. Selvi, "Bypassing HTTP strict transport security," *Black Hat Europe*, vol. 54, 2014.

[16] A. Malhotra, I. E. Cohen, E. Brakke, and S. Goldberg, "Attacking the Network Time Protocol," in *NDSS*. The Internet Society, 2016.

[17] P. Mccorry, "The inactivity leak," https://www.cryptofrens.info/p/the-inactivity-leak, 2023.

[18] J. Kwon, "Tendermint : Consensus without mining," 2014.

[19] W. Y. M. M. Thin, N. Dong, G. Bai, and J. S. Dong, "Formal Analysis of a Proof-of-Stake Blockchain," in *ICECCS*. IEEE Computer Society, 2018, pp. 197–200.

[20] J. Sun, Y. Liu, J. S. Dong, and C. Chen, "Integrating specification and programs for system modeling and verification," in *TASE*. IEEE Computer Society, 2009, pp. 127–135.

[21] J. Sun, Y. Liu, J. S. Dong, and J. Pang, "PAT: towards flexible verification under fairness," in *CAV*, ser. LNCS, vol. 5643. Springer, 2009.

[22] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *SOSP*. ACM, 2017, pp. 51–68.

[23] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure Proof-of-Stake Blockchain Protocol," in *CRYPTO (1)*, ser. LNCS, vol. 10401. Springer, 2017, pp. 357–388.