On the characterization of blockchain consensus under incentives

Sara Tucci

Invited Paper: On the Characterization of Blockchain Consensus under Incentives

Sara Tucci-Piergiovanni[1]

CEA LIST, PC 174, Gif-sur-Yvette, 91191, France

Abstract. One of the novel aspects of blockchains is the intertwining of consensus properties and incentives. An incentive model determines participant behaviors and then the possibility to reach consensus. In this paper we propose a methodological approach to characterize an incentive model for blockchain consensus. An incentive model is defined through the characterization of an oracle, along with its failure model, and blockchain participants behaviors. The oracle assures Safety properties at the expense of Liveness, since Liveness is in the hands of participants that can behave obediently, strategically or in an adversarial way. We then apply the proposed methodology to define and analyze incentive models of popular blockchain solutions. The paper concludes on future research directions that can take advantage of the proposed characterization.

Keywords: Blockchains · Consensus · Incentives models

1 Introduction

Consensus in blockchains means to get an agreement on a unique version of a ledger where each block in the blockchain offers an updated version of the ledger state chained to a previous block. In an ideal blockchain, there is a single sequence of blocks on which participants agree. In contrast to this situation, forks might happen leading to multiple versions of the ledger.

The fairly unique aspect of blockchains is that consensus is driven by incentives. In blockchains an incentive model implements specific financial arrangements for participants influencing their behavior.

To make an example, the maintenance of the ledger in Bitcoin proceeds as follows: participants create blocks including users transactions along with the pointer to the previous block and the solution of a cryptopuzzle; diffuse each newly created block in the network; and store blocks arranged in a tree structure. In Bitcoin, the protocol embeds a rule, which is the longest chain rule, which allows selecting a chain in the tree. This way, blockchain creators chain their newly created block to the selected chain. It is important to highlight that strategic participants does not obey blindly to the longest chain rule but they act strategically selecting the longest chain only if this maximize their utility. Recently, it has been shown that the longest chain rule under the Bitcoin incentive model is the best choice for strategic agents, i.e., the Bitcoin consensus

mechanism along with its incentive model is incentive compatible [9][1]. The Bitcoin's incentive model provides rewards for block creators and takes into account proof-of-work expenditures. We can then state that incentive models and the possibility of reaching an agreement on a common chain are intertwined, therefore, to preserve consensus properties proper incentive models must be defined.

This paper considers the problem of characterization of blockchain consensus and related incentive models. The objective is to capture incentive models for consesus in abstract way, without entering in the detail of specific (often complex) protocols. To this end, we propose a specification of blockchain consensus that extends the abstract formal specification of blockchains proposed by Anceaume et al. in [7]. Anceaume et al. [7] model a blockchain as a concurrent abstract data type that can be concurrently accessed by an unlimited number N of participants. The data type is a rooted tree, called block tree, in which vertices represent blocks, its root corresponds to the genesis block, and each root-to-leaf path is a blockchain. The two supported operations are *append*, which adds a new vertex to the tree, and *read*, which returns a blockchain of the tree selected according to some *fitting function*.

The main feature of [7]'s formalisation is the presence of an *oracle* that helps to append blocks to the tree. Intuitively, the oracle, called by participants during the *append()* operation, controls the maximum branching factor of the block tree. The oracle allows a vertex in the tree to have up to some parameter k children. In [7] it has been shown that strong consistency, i.e., no-forks, can be achieved only assuming an oracle with branching factor $k = 1$ and that this oracle is at least as strong as Consensus. The relationship within blockchain consistency and consensus has been then formally established. The branching factor of an oracle, moreover, fully characterizes a blockchain, where an oracle with $k = 1$ characterizes blockchains resorting to consensus and $k > 1$ characterizes consensus-free blockchains. In other terms, the essence of any blockchain is captured by the assumed oracle, which is a simple abstraction.

In this paper we then characterize incentive models by augmenting oracles (not the block tree) with costs and rewards associated to its execution. In this respect we will focus on characterization of consensus-based blockchains, i.e. blockchains using an oracle with branching factor equal to 1. Let us note that since consensus cannot run among an unbounded number of processes, current solutions, e.g. [19, 21], attempt to solve the problem through rotating committees of fixed size n. In this case the production of a single block needs coordination among the n committee members and rewards are distributed among members once the block is produced. The oracle then needs to feature a selection phase: for each height of the chain n committee members out of the total number of participants $N \geq n$ are selected, members that need to agree on the next block. After the selection phase, a voting phase among the n members starts.

[1] Incentive compatibility can be established considering different solution concepts, such as dominant strategies and Nash equilibria. Incentive compatibility of Bitcoin has been shown assuming Nash equilibria as solution concept. Formal definitions of these notions will be presented later in the paper.

We abstract the voting phase considering a function f that maps votes on a set of proposals to a joint decision. Since participants may decide to vote or not and voting has a cost, we define a quorum ν, the minimum number of votes necessary to trigger a decision. Once the voting phase completes, if a valid decision is reached, the oracle distributes rewards to participants.

The incentive model characterization is then completed by proceeding to the following steps: (1) definition of the failure model of the oracle – the oracle can be either correct or it can lose/corrupt votes, (2) definition of the type of participants, which can obedient, strategic or adversarial and (3) definition of the rewarding function, i.e., how rewards are distributed to the committee.

Different incentive models capturing current popular blockchain solutions are then analyzed. Incentive models are analyzed exploring their influence on Liveness (eventually a block is decided) only. Safety properties are indeed always guaranteed by the oracle at the expense of Liveness, driven by the whole incentive model. The paper concludes highlighting that the quest for liveness-preserving incentive models is in its early days.

The paper is organized as follows. Section 2 introduces blockchain oracles previously presented in [7]. Section 3 characterizes incentive models for consensus-based blockchains. Section 4 presents analyses of incentive models of current popular solutions and finally Section 5 discusses research directions.

2 Blockchain core abstractions

Anceaume et al.[7] present an abstract and formal characterization of blockchains that allows to capture their properties independently from protocol-specific mechanisms (Proof-of-Work, Proof-of-Stake, Consensus, etc) and system model assumptions (synchrony, type of faults, number of participants). The main feature of the characterization is the presence of so-called blockchain oracles that (i) abstract the validation process of blocks in the chain, (ii) regulate forks and (iii) abstract the communication medium.

In this section we briefly present this characterization, but for readability we omit all the formal notations giving only the essential concepts.

2.1 The blocktree and oracle models

Following [7], any blockchain is characterized by a concurrent data strucuture representing the blockchain and the type of oracle that helps maintaining it, specified as follows.

The blockchain tree. A blockchain is a direct rooted tree bt that can be accessed by $read()$ and $append(b)$ operations. Each vertex of the tree is a block and any edge points backward to the root, called genesis block and denoted as b_0. The height of a block refers to its distance to the root.

The read operation $read()$ returns a branch of the tree, from the genesis to one leaf, selected through a so-called fitting function $f(bt)$, which is protocol specific.

Each walk, or chain, in the tree as has an associated score that is monotonically increasing w.r.t. inclusion of blocks.

The *append*(b) operation takes one parameter, the block to be appended. To be appended a block must be valid. The validity of a block is protocol-specific. The *append*(b) operation returns true if the block is appended to the tree, false otherwise. If the block has been appended it is valid.

Concurrent specification. Both append and read operation can be invoked concurrently by blockchain participants. Under concurrency, the data type satisfies Block Validity: Blocks in the blockchain returned by the read operation are valid with respect to a validity predicate; Local monotonic read: The score of the blockchain returned by subsequent reads from the same process is monotonically nondecreasing; and the Ever-growing tree property: The score of the returned blockchain eventually grows. As for consistency, two alternative consistency criteria are defined, namely: (i) *Strong consistency*: In addition to the above properties, for any two blockchains returned by a read operation, one is a prefix of the other; or alternatively (ii) *Eventual consistency*: In addition to the above properties, if a blockchain with score s is returned, then at most a finite number of read operations return blockchains that do not share the same prefix up to score s.

Strongly consistent blockchains do not allow forks, while eventually consistent blockchain admit occurrences of forks that are eventually solved.

Blockchain Oracles. Oracles are generic modules able to abstract the generation of valid blocks, the communication medium and to regulate the branching factor of the block tree. During the *append*(b) operation, the oracle is called through a $getValidBlock(b^*, b)$ operation that takes two parameters, the proposed block b and a block of the block tree b^*, chosen as proposed parent of b. Let us note that b^* is freely chosen by the invoking process and that the block b is valid only if the $getValidBlock(b^*, b)$ operation successfully terminates. It is assumed that only the oracle is able to make the block valid, i.e. to implement the $getValidBlock(b^*, b)$ operation.

The oracle releases valid blocks depending on the merit parameter $\alpha_i \in [0, 1]$ of the invoking process i. In [7], for each α_i the oracle endows an infinite tape $tape_{\alpha_i}$ of elements in the set $\{\top, \bot\}$. Each time the getValidBlock is invoked by a process with merit α_i, an element is popped by $tape_{\alpha_i}$. If the popped element is $\top$, then the valid block is released, otherwise the operation returns $false$. In this deterministic version[2] the oracle, if invoked infinitely often by i with $\alpha_i > 0$ with the same pair of blocks (b^*, b), will eventually release a valid block guaranteeing a form of fairness. In this paper, for sake of generality and to ease the presentation we assume the presence of a higher-level function $select(\alpha_i)$ that accesses the tape and returns true if the popped element is $\top$. The pseudo-code of the getValidBlock operation is shown in Algorithm 1.

[2] Even if oracles defined in [7] are deterministic, probabilistic versions can be easily derived by associating a probability to pop $\top$ proportional to the merit.

Algorithm 1 `getValidBlock` operation at b^*

```
1: upon ⟨getValidBlock(b*, b_i)⟩ from process i with α_i do
2:   if (select(α_i)) then
3:     validBlocks = validBlocks ∪ {b_i}
4:     return b_i
5:   else
6:     return false
```

Algorithm 2 `setValidBlock` operation at b^*

```
1: upon ⟨setValidBlock(b*, b_i)⟩ from process i do
2:   if (b*.children.size < k) ∧ (b_i ∈ validBlocks) ∧ (b_i ∉ b*.children) then
3:     b*.children ← b*.children ∪ {b_i}
4:   return b*.children
```

Once a process gets a valid block from the oracle, it can now set it through a $setValidBlock(b^*, b)$. As soon as the setValidBlock operation returns we have two cases: if b is returned then $append(b)$ returns true, otherwise $append(b)$ returns false. It is possible that more than one participant gets a valid block to append to b^*, in that case the oracle regulates forks depending on its *branching factor* k. If the oracle has a bounded branching factor, then the oracle guarantees that the returned set of any successfully executed $setValidBlock(b^*, b)$, returns a set of children of bounded size k containing successfully set blocks, then possibly not containing b. If k is unbounded, then the oracle returns a set of children containing b. The pseudo-code of the setValidBlock operation is shown in Algorithm 2. In case of unbounded branching factor, the pseudo-code is either modified by removing the first term of the *if* condition or by considering $k = \infty$.

In [7] it has been shown that (i) the oracle with $k = 1$ is equivalent to Consensus and (ii) Consensus is necessary for strongly consistent blockchains.

2.2 Oracles, a closer look

Oracles, as already mentioned, abstract away the consensus mechanism used in blockchains. Let us to have a closer look at the oracle characteristics. The getValidBlock operation abstracts away the validation process of blocks. For instance, if we think to the proof-of-work mechanism, we can see the getValidBlock invocation as a query to an oracle that gives to the process the solution of the proof-of-work or false otherwise.

The setValidBlock operation, has a subtler role. Let us take our weakest oracle, with $k = \infty$. The setValidBlock operation only means in this case to make the block *visible* in the blockchain. Once the setValidBlock returns, the block is accepted by the blockchain, i.e., it is appended. We can think to the set operation as a successful write of a new block in the blockchain state. It is important to stress out that actual update of a state change is far from being trivial in a open peer-to-peer network, subject to participant churn. For that

reason, it is extremely useful to resort to oracles. In this perspective, all the blocks returned by setValidBlock operations can be viewed as stable vertexes in the tree: if some processes sets a given block b, than successive setValidBlock will report b in the returned set[3].

The shared burden between oracles and blockchain participants. Oracles, by definition, are local to one particular vertex of the tree: each participant chooses to call the getValidBlock on the parent vertex of her choice, and to call setValidBlock to try to append the block. This suggests that it is up to participants to guarantee the consistency of the blockchain and not to oracles. For instance, if there is just one participant in the blockchain that keeps calling the oracle to append a block to the genesis one, and the oracle has $k = \infty$, the ever-growing tree property will never be guaranteed. At some point the process must jump to a leaf to make the tree growing. This is also true by assuming, for the same scenario, an oracle with $k = 1$: the calling participant will always terminate her append operation by returning false, but the tree will stop growing. In case of weaker oracles with $k > 1$, if there are two processes jumping alternatively and concurrently on two parallel branches of the tree, it is up to them to eventually select and read the same chain to guarantee eventual consistency and repair forks. The separation of duties between oracles and blockchain participants described so far is particularly meaningful when strategic behavior is assumed. Blockchain participants thanks to the oracle have the possibility to update the block tree, but it is up to them to choose the chain they prefer when reading and appending; this choice is done strategically. In this respect it is interesting to highlight that stronger oracles reduce the strategic space with respect to weaker ones, but even assuming our strongest oracle with branching factor $k = 1$, strategic choices are relevant to guarantee progress.

2.3 Enriching oracles with incentives

In all the implementations employing the proof-of-work, a cost must be associated to block validation. For those implementations exempted by the proof-of-work, to avoid spamming – a process maliciously sending too many blocks – either a mini proof-of-work is employed or a so-called slashing mechanism allows to burn some coins of the malicious process [19, 14]. In all cases we can abstract a cost associated with the generation of valid blocks. This cost can be allocated to the process either at the end of the invocation of the getValidBlock or when then process invokes the setValidBlock operation – this is to take into account a cost only when a block is sent in the system.

As for rewarding, current solutions can roughly be divided in two families: those where only one participant is accountable for the appended block and those where more than one participant is accountable for it. Rewarding is then given to accountable participants for the block. We are interested in the second family

[3] In [7] the consumeValidBlock, called in [7] consumeToken operation, has been reduced to the Generalized Lattice Agreement abstraction [12].

where the oracle has branching factor $k = 1$, i.e., an oracle implemented through Consensus in a deterministic setting. In current solutions based on Consensus [19, 21] a block is created coordinately by committees of n participants selected out of N blockchain participants and the reward is shared among committee members. We will define an incentive model for this class of blockchains. We will introduce all the elements needed to define a so-called Consensus Incentive Model: (i) an adaptation of the oracle with branching factor $k = 1$ to model selection and rewarding, (ii) a failure model for the oracle, (iii) participants characterization in terms of their type (obedient, adversarial, strategic) and behavior.

3 Consenus Incentive Model definition

Oracle Characterization. The oracle is an oracle with branching factor $k = 1$ called hereafter Consenus oracle, with a cost associated to the invocation of setValidBlock. The oracle, as current solutions, e.g. [18, 19, 14, 11], selects committee members that have most merit and share the reward among committee members, even though more sophisticated models have been devised[4]. The oracle is specified as follows (Algorithm 3 and 4). The oracle when invoked at a given block b^* through $getToken(b^*, *)$ grants valid blocks up to n invokers, by following a given selection criterion and puts the process i in a validator set. Once the process i gets a valid block it may decide to try to append the block in the blockchain, through $setValidBlock(b^*, b)$. The operation starts a voting phase to collect proposals. When the phase ends, only if $\nu \leq n$ proposals are collected a decision is taken through a function f that maps proposals to the block to be set. Note that the duration of a voting phase depends on the system model assumed. In synchronous systems the voting phase lasts at least the time required to gather the ν proposals, but it could last more and gather more than ν proposals. In case ν proposals are gathered, but the decision is not valid the oracle returns $false$. The reward function takes as parameter the validator set, the proposal and the decision. If all the validators are rewarded no matter if they voted or not, then we have a *reward all* rewarding scheme; alternatively, if only voters get the reward, we have a *reward only voters* scheme.

Failure models of the oracle. The oracle can be:

- *correct*: The pseudo-code (Algorithm 3 and 4) is correctly executed.
- *vote corruption*: the proposals set at line 7 can contain up to c non-valid votes, because of a failure of the valid block test.
- *vote omission*: in the proposals set at line 7 up to m votes can be lost.

The oracle is not a strategic entity, the failure model assumed abstracts away possible system failures. The *reward* function is always executed correctly.

[4] Some implementations [14, 11] provide a delegation mechanism in which a participant can delegate its merit to another one, in this case the reward is shared among delegators as well.

Algorithm 3 Consensus Oracle with Incentives: getValidBlock operation at b^*

```
1: upon ⟨getValidBlock(b*, b_i)⟩ from process i with α_i do
2:   if ((select(α_i) ∧ (n ≤ j)) then
3:     validBlocks = validBlocks ∪ {i}
4:     validatorSet ← validatorSet ∪ {b_i}
5:     j + +
6:     return b_i
7:   else
8:     return false
```

Algorithm 4 Consensus Oracle with Incentives: setValidBlock operation at b^*

```
1: upon ⟨setValidBlock(b*, b_i)⟩ from process i do
2:   if (b*.children.size < 1) ∧ (b_i ∈ validBlocks) then
3:     if voting phase not started then
4:       start voting phase
5:     b*.proposals ← b*.proposals ∪ {b_i}

6: upon ((end voting phase) ∧ (proposals.size = ν)) do
7:   decision ← f(proposals)
8:   reward(validatorSet, proposals, decision)
9:   if decision ∈ validBlocks then
10:    return b*.children
11:  else
12:    return false
```

Participants Characterization.

Participants action space. Each participant has a strategy s_i

$s_i \in \{0, 1, \perp\}$, where 1 maps to the decision of sending a vote, 0 of not sending a vote, and $\perp$ of sending an invalid vote.

Types of Participants. We define then three types of participants:

- *strategic*: a player that chooses the strategy that maximizes her payoff
- *obedient*: a player whose strategy is fixed to 1 under the condition that utility is greater than a given threshold γ. In other terms the obedient participant has limited resources, needing then some degree of fairness [15, 5].
- *adversary*: a player whose strategy is either 0 or $\perp$.

Later we will extend our oracle to a *multi-ballot oracle*, to include in the action set the action of sending two different votes in two different ballots in the same voting phase. This is to include the classical Byzantine misbehavior of sending different messages to different processes. This extended action set will be discussed for solutions based on BFT consensus.

Properties to guarantee. Properties to guarantee at each height of the chain are traditional Consensus properties: Agreement (at most one decision is taken), Validity (the decision value must be valid) and Termination (eventually a decision is taken). Agreement and Validity are safety properties while Termination

is a Liveness property. In our proposed oracle construction participant behaviors can only hinder Liveness, leading to never return a valid block (line 10). This is easy to see if the oracle is assumed correct. In case of votes corruptions (modeling a vote for an invalid block), the decision function f is able to filter out corrupted proposals. If no valid proposal can be chosen, the f returns $\perp$ and the block is not returned.

The agreement property is by construction not threatened by the given action set, but easily guaranteed with a multi-ballot oracle capturing PBFT solutions as we will see later.

The main principle is to have an oracle assuring Safety at the expense of Liveness and to study how participants behavior under the assumed incentive model influences Liveness.

Utility functions. Before being able to establish participant strategies we need to define utility functions. A utility function depends on the oracle failure model and the action set. The simplest utility function, defined assuming a correct oracle, is as follows.

$$u_i = \Sigma_{\kappa=0}^{n_i^r} R_i^\kappa - \Sigma_{\ell=0}^{n_i^v} C_i^\ell \quad (1)$$

where

- n_i^r is the number of setValidBlocks operations successfully executed and such that i belongs to $validatorSet$ (reward all) or i's proposal belongs to $proposals$ (reward only voters) at line 8.
- n_i^v is the number of setValidBlocks operations successfully executed and such that i voted, i.e., i's proposal belongs to $proposals$ at line 5.

Weaker oracles can also envisage punishments. For instance an oracle suffering from vote corruptions, can contain in the proposal set some invalid blocks. We consider that the decision function f could in this case select an invalid block but that the decision is not returned, leading to Liveness violation. The participants, if an invalid block is decided or even proposed, can be punished.

Participants strategies. Once the utility function is in place participant strategies can be defined.

Strategies of obedient participants correspond to the expected behavior from the designer point of view. The strategy under our oracle models and action space is then naturally 1 as long as the participant can pay the inflicting costs. For obedient participants it is assumed that the participant enters the system with an endowment and that such endowment is modified solely by costs and rewards defined by her utility function. The threshold γ must be set to make sure that the participant is rewarded often enough to not exhaust her resources. In the weakest model γ is greater than zero.

Strategies for adversarial players can be determined considering strategies that undermine properties to guarantee, i.e. Termination. It assumed that the utility function is known by adversarial processes.

Strategies for strategic participants can be determined in the framework of game theory. Game theory focuses on predicting individual players' strategy and payoffs. For each combination of players and possible strategies, there is a payoff. Game theory analyzes which strategies strategic players will play in the game.

Given a game, it is interesting to look for dominant strategies. A dominant strategy for a player is a strategy leading to the best payoff, no matter how other players may play. However, dominant strategies there not always exist.

Nash Equilibrium is an equilibrium where each player's strategy is optimal given the strategies of all other players. A Nash Equilibrium exists when no player would take a different action as long as every other player remains the same. Nash Equilibria are self-enforcing; when players are at a Nash Equilibrium they have no desire to deviate, otherwise they will be worse off. Interestingly, given any finite game a Nash equilibrium there always exists.

More formally a Nash equilibrium can be defined as follows:

Nash equilibrium. Let $(\mathcal{S},\mathcal{U})$ a game with n players where $\mathcal{S}_i$ is the strategy set for the agent, $\mathcal{S} = \mathcal{S}_1 \times \ldots \times \mathcal{S}_n$ is the set of strategy profiles and $\mathcal{U}$ is the set of utility functions $\mathcal{U} = u_1 \times \ldots \times u_n$ mapping a strategy $s_i \in \mathcal{S}_n$ to a payoff $u_i(s_i)$.
A strategy $s_i \in S_i$ for the agent i is a mapping from each state of the game to an action in the action space of the agent. A strategy tells a player what to do for every possible state of the game throughout the game[5]. When each agent chooses a strategy from its strategy set we get a strategy profile $s = (s_1, \ldots, s_n) \in \mathcal{S}$.
Let us denote with $(s\,|^i, s_i^*)$ the fact that i deviates from s by doing $s_i^* \in S_i$.
A strategy profile s is a pure Nash Equilibrium if and only if for each i, and for all strategies $s_i^* \in S_i : u_i(s\,|^i, s_i^*) \leq u_i(s)$.

In the following we will analyze incentive models with strategic players under the Nash equilibrium solution concept.

Failure models of participants. We assume that obedient and strategic participants are able *to correctly execute their strategy*, i.e. they are correct processes with respect to their strategy. We also assume that adversarial participants can execute their strategy in the worst case, but they can be affected by unexpected failures, i.e. they are Byzantine processes[6]. Note that our model separates failures from strategies (that could deviate or not from designer's one). This separation, inspired by [13], defines a slightly different model than the BAR model [2]. Our obedient participants are neither altruistic (they do not maximize the benefits of

[5] A strategy can be viewed as an algorithm. The state in game theory is called the information set that is evaluated each time it is updated, to select the next action or move.

[6] Even if Byzantine failures are defined as arbitrary deviations from the prescribed behavior, an adversarial argument is assumed to prove protocols under Byzantine processes. This way the strategy of the Byzantine participant is determined.

others in the general case) nor correct since they have limited resources. Moreover, any strategic player is assumed correct with respect to her chosen strategy (on the contrary the participant must be assumed Byzantine).

Efficiency. Any incentive model can be analyzed under the efficiency point of view. Given ν the threshold to produce a block, efficiency is the ratio between ν and the number of votes determined by chosen strategies.

4 Analysis of current solutions

In this section we illustrate first how to capture current solutions under an incentive model and then we analyze the incentive model itself. The section has a pedagogical purpose and let us observe how assumptions on the particular combination of oracle failure model, participants types and the reward schemes impact Liveness. We analyze two reward schemes: *reward only voters* and *reward all.*

Strategic incentive models for synchronous leader-based solutions in a one shot game. In solutions like [11][14], a single leader is elected for each height of the chain. The leader sends its block and the $n-1$ other processes send a vote for the block. We can abstract these solutions by our oracle without failures, n participants and a decision function f that selects the proposal of the leader where all the proposals are votes. We need for the block to be produced at least ν votes, otherwise liveness will not be guaranteed. Assuming a synchronous system means to collect at the end of the vote phase all the votes to potentially reward them (the oracle is assumed correct, no message losses occur). We assume n strategic participants, utility function (1) for only one height of the tree with $R > C$, assuming a one shot game. Strategies of participants are as follows [3].

Reward only voters. For the reward only voters scheme we have two cases: $\nu > 1$ and $\nu = 1$. The case of $\nu > 1$ has two Nash equilibria. In the first equilibrium all the participants send a vote, i.e., they call the setValidBlock operation. In the second equilibria nobody send a vote; violating Liveness. In the case of $\nu = 1$ we have a single Nash equilibrium where all n participants vote. The efficiency of the mechanism is $\frac{\nu}{n}$.

Reward all. For the reward all mechanism, we have two cases: $\nu > 1$ and $\nu = 1$. The case of $\nu > 1$ has multiple Nash equilibria, where either ν participants sends a vote or nobody sends a vote; violating Liveness. In the case of $\nu = 1$ we have a n Nash equilibra, in each equilibrium, exactly one process sends a vote. Efficiency is optimal.

Note that a good Nash equilibrium for both schemes is reached when $\nu = 1$, because the participant is pivotal: in the strategy profile where nobody send the vote, she will be better off by deviating, i.e. sending the vote. This is true for all the participants if only voters are rewarded. In the reward all scheme, this is true for ν participants.

Comparing the two mechanisms the *reward all* is more efficient, in the sense that less money is distributed to participants in good equilibria. Both of them, however, if $\nu > 1$ have bad equilibria resulting in a coordination failure and liveness violation.

Adversarial-strategic incentive model for synchronous leader-based solutions in a repeated-game. In [3] a more complex model has been considered, where a combination of adversarial and strategic players is assumed and more leaders can be elected in a round-robin fashion to produce a block at a given height of the chain. The oracle used is an oracle accepting invalid blocks but able to filter invalid decisions out. The reward scheme assumed is a *reward only voters* with a cost inflicted to all the committee members if an invalid block is selected by the decision function f, i.e. if the leader proposed an invalid block that has been accepted by ν participants. The study assumes as well that participants have the possibility to check upfront if the leader's proposal is valid or not at some additional cost. Moreover, it is assumed that the reward is greater than the inflicted cost and the cost of accepting an invalid block is greater than the reward (see [3] for the utility function formal definition).

It is easy to see that the adversarial strategy to inflict the maximum damage to other participants is to propose an invalid block and hope to get a sufficient number of votes to have the block accepted. In this case all of them will be penalized and Liveness threatened.

As for strategic players, strategies have been found considering multiple voting rounds (for the same height of the chain) where for each round a leader is chosen in a round robin fashion. The study finds multiple Nash equilibria (we encourage the reader to look at the entire paper). Among many bad equilibria leading to coordination failure and liveness violation, the rewarding mechanism in the proposed model shows a good equilbrium if $\nu > a$ where a is the upper bound on adversarial participants. In good equilibrium invalid blocks (proposed by adversarial players) are rejected, while valid blocks (proposed by strategic players) are accepted. This implies that, if round $t = a + 1$ is reached, the players know that during all the previous a rounds the proposers were Byzantine (to draw this inference, the strategic players use their anticipation that all participants play equilibrium strategies). Consequently, at round $a + 1$, the proposer must be strategic, and all players anticipate the proposed block is valid. So, no strategic player needs to check the validity of the block but all send a vote, which brings them an expected gain equal reward minus the cost to send a vote. This is larger than their gain from deviating by not sending a message or by checking the block.

Obedient-adversarial incentive models for BFT blockchains Blockchains as [21, 19] use BFT Consensus protocols, e.g. [22, 4, 6], inside committees with n processes where at most $f = \lfloor \frac{n-1}{3} \rfloor$ participants are Byzantine faulty. The other $n - f$ processes are assumed correct.

For this class of applications, the model assumed is abstracted by our Consensus oracle with f adversarial participants, $n - f$ obedient participants and $\nu = n - f$. These solutions are leader-based, then multiple voting phases can be activated with leaders selected in a round-robin fashion. The eventual synchrony assumption is dealt with an adaptive timeout meant to catch the unknown message delay eventually on some voting phase. We abstract this behavior assuming that at the end of a single voting phase all sent votes are collected, but the oracle can lose at most one third of sent votes. After an finite but unknown number of voting phases, the oracle ceases to lose messages. To capture these solutions we need as well to extend our action set, to let an adversarial participant to behave like a Byzantine process sending two different votes to different processes, instead of one single vote. To this aim the oracle can be extended to a n ballot voting phase where each participant votes in n different ballots, one for each participant i. Each ballot has n entries indexed by the participant identifier and ballots are filled up to at least ν votes. This way an adversary j can vote different blocks in different ballots. The decision function f takes the first entry that have two third of common valid values. Let us to make an example. Process $p1$ is Byzantine, while $p2, p3, p4$ are correct. Values proposed on the n ballots are as follows $p1 = \{1, 1, 0, 1\}$, $p2 = \{2, 2, 2, 2\}$, $p3 = \{3, 3, 3, 3\}$, $p4 = \{4, 4, 4, 4\}$. The oracle can lose at most one vote per ballot. Values lost by the oracle are the values proposed by pi for the $(i+1)^{th}$ ballot. The ballots state, where $\{\}_i$ is the i^{th} ballot, at the end of the vote phase is as follows: $\{1, 2, 3, \perp\}_1$, $\{\perp, 2, 3, 4\}_2$, $\{0, \perp, 3, 4\}_3$, $\{1, 2, 3, \perp\}_4$. The oracle takes a decision that must be the same for obedient participants, without knowing who the Byzantines are. The oracle takes the common value contained in the first entry of two-thirds of ballots (if any), then 2 in the scenario above, and notifies all about the decision. We say that p_1, p_2 and p_4 voted for the decided value because their corresponding ballot contained the decided value. Note that "more than two-thirds correct" processes assumption neutralizes the effect of the byzantine misbehavior, i.e. in these conditions the decision value there always exists. Under these conditions, the adversarial behavior of a Byzantine will then try to hinder the participation of obedient participants to violate Liveness (to lower the threshold of correct processes). This behavior depends on the rewarding scheme. Let us recall that unlike a correct process an obedient one has limited resources, dictated by her endowment. An obedient process must be rewarded often enough to maintain a balance greater than zero. If not, the process with no endowment will not be able to cover the cost of sending her votes. Let us consider rewarding as follows. *Rewarding the processes that voted the decided value.* In the above mentioned scenario if the oracle does not lose the vote of the Byzantine, then the Byzantine gets a reward and $p3$ does not. $p3$ can be excluded from the reward also in the case the Byzantine sends the same valid value, but the oracle loses the first entry of $p3$. If these scenarios repeat too long, $p3$ can exhaust her resources and will not be able to vote anymore. The best strategy for the adversarial participant to hinder Liveness is then to participate with valid values hoping $p3$ reaches endowment 0 and then to stay silent forever.

Rewarding all. This scheme gives a reward to all the processes, even those that do not send her vote. Note that this is the safer method here, even if less efficient, because a Byzantine has no power to make the obedient worse off.

5 Discussion and Research Directions

The proposed characterization highlighted the interlacement of participants' strategy and rewarding schemes in various incentive models based on oracles with branching factor equal to 1. Recent research [20] is now focusing on BFT consenus to see how to increase the threshold f of corruption faults. In this direction, some effort is spent in detecting corrupted processes [10] because this capability could allow to exclude them from the reward, or to punish them, hoping to tolerate the presence of more than one third of adversarial processes. As a recommendation it is important not to sacrify liveness for efficiency here and to clearly state the incentive model under study. In our characterization, for instance, an adversary is punishment-insensitive. An interesting research question would be to characterize adversarial participants sensitive to rewarding/punishment by assuming limited resources for them. Another interesting line of research is to consider preferences about transactions contained in blocks. Approaches that explored leader election [1] and consensus [16, 8] under these assumptions could be analyzed under our incentive models. Finally, obedient models could be refined to consider more sophisticated fairness-related concepts and the interlacement between merit and participant endowment [15, 5, 17]. We hope that the provided characterization can help in exploring these research directions.

Acknowledgments. This position paper assembles ideas and results emerged through research conducted with E.Anceaume, A. del Pozzo, M. Potop-Butucaru and O. Gurcan. A special thanks goes to Y. Amoussou-Guenou working hard in this middle earth between distributed computing and economy and to Prof. Bias that literally opened us the door to economic methodologies.

References

1. Abraham, I., Dolev, D., Halpern, J.Y.: Distributed protocols for leader election: A game-theoretic perspective. ACM Trans. Economics and Comput. **7**(1), 4:1–4:26 (2019)
2. Aiyer, A.S., Alvisi, L., Clement, A., Dahlin, M., Martin, J.P., Porth, C.: Bar fault tolerance for cooperative services. In: Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP'05). pp. 45–58 (2005)
3. Amoussou-Guenou, Y., Biais, B., Potop-Butucaru, M., Tucci-Piergiovanni, S.: Rationals vs byzantines in consensus-based blockchains. CoRR **abs/1902.07895** (2019)
4. Amoussou-Guenou, Y., Del Pozzo, A., Potop-Butucaru, M., Tucci-Piergiovanni, S.: Dissecting tendermint. In: NETYS 2019

5. Amoussou-Guenou, Y., Del Pozzo, A., Potop-Butucaru, M., Tucci-Piergiovanni, S.: Correctness and Fairness of Tendermint-core blockchain protocols. Research report (2018), https://hal.archives-ouvertes.fr/hal-01790504
6. Amoussou-Guenou, Y., Pozzo, A.D., Potop-Butucaru, M., Tucci-Piergiovanni, S.: Correctness of tendermint-core blockchains. In: 22nd International Conference on Principles of Distributed Systems, OPODIS 2018, December 17-19, 2018, Hong Kong, China. pp. 16:1–16:16 (2018)
7. Anceaume, E., Del Pozzo, A., Ludinard, R., Potop-Butucaru, M., Tucci-Piergiovanni, S.: Blockchain Abstract Data Type. SPAA 2019 (2019)
8. Bei, X., Chen, W., Zhang, J.: Distributed consensus resilient to both crash failures and strategic manipulations. CoRR **abs/1203.4324** (2012)
9. Biais, B., Bisière, C., Bouvard, M., Casamatta, C.: The blockchain folk theorem. The Review of Financial Studies (2019)
10. Civit, P., Gilbert, S., Gramoli, V.: Polygraph: Accountable byzantine agreement. IACR Cryptology ePrint Archive **2019**, 587 (2019)
11. David, B., Gazi, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II. pp. 66–98 (2018)
12. Falerio, J.M., Rajamani, S.K., Rajan, K., Ramalingam, G., Vaswani, K.: Generalized lattice agreement. In: ACM Symposium on Principles of Distributed Computing, PODC '12, Funchal, Madeira, Portugal, July 16-18, 2012. pp. 125–134 (2012)
13. Feigenbaum, J., Shenker, S.: Distributed algorithmic mechanism design: recent results and future directions, distributed computing column. Bulletin of the EATCS **79**, 101–121 (2003)
14. Goodman: Tezos: a self amending crypto ledger. https://tezos.com/static/white_paper-2dc8c02267a8fb86bd67a108199441bf.pdf
15. Gürcan, Ö., Del Pozzo, A., Tucci-Piergiovanni, S.: On the bitcoin limitations to deliver fairness to users. In: On the Move to Meaningful Internet Systems. OTM 2017 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2017, Rhodes, Greece, October 23-27, 2017, Proceedings, Part I. pp. 589–606 (2017)
16. Halpern, J.Y., Vilaça, X.: Rational consensus: Extended abstract. In: Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016. pp. 137–146 (2016)
17. Karakostas, D., Kiayias, A., Nasikas, C., Zindros, D.: Cryptocurrency egalitarianism: A quantitative approach. Tokenomics International Conference on Blockchain Economics, Security and Protocols 2019 (2019)
18. Kwon, J., Buchman, E.: Cosmos: A Network of Distributed Ledgers. https://cosmos.network/resources/whitepaper (visited on 2018-05-22)
19. Kwon, J., Buchman, E.: Tendermint. https://tendermint.readthedocs.io/en/master/specification.html (visited on 2018-05-22)
20. Malkhi, D., Nayak, K., Ren, L.: Flexible byzantine fault tolerance. ACM CCS (2019)
21. Various: The Libra Blockchain. https://developers.libra.org/docs/assets/papers/the-libra-blockchain.pdf
22. Yin, M., Malkhi, D., Reiter, M.K., Golan-Gueta, G., Abraham, I.: Hotstuff: BFT consensus with linearity and responsiveness. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019. pp. 347–356 (2019)