Eiffel: Extending Formal Verification of Distributed Algorithms to Utility Analysis

Arian Baloochestani, Leander Jehl
Department of computer science and electrical engineering
University of Stavanger
Stavanger, Norway
arian.masoudbaloochestani@uis.no, leander.jehl@uis.no

***Abstract*—The security of distributed systems, such as blockchains, relies heavily on the active participation of sufficient participants within the network. In order to incentivize such participation, different blockchains implement different rewarding mechanisms. However, for these mechanisms to be effective, it is essential that they are fair and accurately implemented, taking into account the rational participants. Consequently, utility analysis plays a vital role in assessing the efficacy of these mechanisms. Formal verification methods, such as TLA$^+$, are commonly employed to ensure the correctness of distributed systems, including blockchains. However, the application of these methods for utility analysis remains relatively unexplored.**

In this paper, we propose Eiffel, a novel approach that extends the formal verification of distributed systems to encompass utility analysis. By leveraging TLA$^+$, Eiffel enables the analysis of rewarding mechanisms and the detection of potential attacks. Eiffel focuses explicitly on the concept of Nash equilibrium, a fundamental notion in game theory that holds significant importance in utility analysis. Through the use of Eiffel, we demonstrate its versatility with different specifications, highlighting its applicability in committee-based blockchains and the Tendermint consensus algorithm. Our analyses show the effectiveness of Eiffel in evaluating rewarding mechanisms and identifying potential vulnerabilities.

***Index Terms*—Formal verification, TLA+, Game theory, Nash Equilibrium,**

I. Introduction

Various industries have adopted blockchain technology due to its ability to enable decentralization and transparency through a shared and immutable ledger [1]. Consensus algorithms play a critical role in blockchain networks by facilitating agreement among distributed nodes. Proof-of-Work (PoW) and Proof-of-Stake (PoS) are the most popular consensus algorithms used by various blockchains. However, they suffer from drawbacks such as resource consumption and security vulnerabilities [2], [3]. Alternatively, some protocols use Byzantine Fault Tolerance (BFT) protocols [4]. These blockchains, referred to as the *committee-based blockchains*, delegate block creation and validation responsibilities to a selected committee of nodes, offering advantages such as scalability, energy efficiency, and enhanced security. Among the committee-based blockchains, Tendermint [5] has become a popular solution due to its robust and efficient consensus algorithm.

The security and performance of consensus algorithms directly depend on the active participation of a sufficient number of participants. Therefore, different rewarding mechanisms are designed to incentivize participation. This is especially true for committee-based blockchains, which rely on the participation of a correct majority- The reward mechanisms need to be fair, meaning that the reward distribution has to be proportional to the participants' work contribution [6]. If a protocol rewards all participants equally regardless of their merit, rational processes skip some parts of the protocol. This problem, known as *free riding*, eventually hurts the protocol security.

While formal verification methods, such as TLA$^+$ [7], are commonly used to abstract and verify distributed systems, their application to analyzing rewarding mechanisms in blockchains remains relatively unexplored. Instead, alternative methods, including game-theoretic notions, are employed to analyze utility and rational behavior in such systems. *Nash equilibrium* is a fundamental concept in game theory that represents a stable state in which no player has an incentive to unilaterally deviate from their chosen strategy, given other players' strategies. The concept of Nash equilibrium is especially important in designing reward mechanisms. If a protocol's correct behavior is a Nash equilibrium, then it is in the best interest of rational processes to follow the protocol since deviation results in losing reward.

In this paper, we propose Eiffel, a novel approach that is added as an extension in formal verification of distributed systems to enable utility analysis. By leveraging TLA$^+$, Eiffel enables the verification of Nash equilibrium within rewarding mechanisms. Our approach employs a dual instantiation method, where two different instances of a given specification are compared. In one of the instances, all processes entirely follow the protocol, while in the other one, a rational process deviates. Using an invariant, Eiffel compares the utility of the rational process in both instances and, therefore, identifies potential Nash equilibria.

To demonstrate the applicability of Eiffel, we present a simplified abstraction of committee-based blockchains, where a selected committee utilizes a BFT protocol [4] protocol to achieve consensus. We apply Eiffel to analyze different rewarding mechanisms within this context. Furthermore, we validate the versatility of Eiffel by applying it to an existing specification of the Tendermint consensus algorithm [5], showcasing how Eiffel can be adapted to different TLA$^+$ specifications with minimal modifications. This is important

since there are several TLA$^+$ specifications available for well-known consensus algorithms such as PBFT [8], HotStuff [9], and Tendermint [10], [11].

In summary, we make the following contributions:

- We introduce Eiffel, a novel dual instantiation method that enables utility analysis in distributed systems using TLA$^+$.
- We demonstrate the application of Eiffel in verifying Nash equilibrium, a crucial notion in utility analysis.
- We apply Eiffel to analyze different rewarding mechanisms within committee-based blockchains using a simplified specification.
- We extend an existing Tendermint specification by incorporating rewarding mechanisms and apply Eiffel to detect potential attacks, such as free riding.
- We showcase the effectiveness of Eiffel in enhancing the security and performance of distributed systems through utility analysis.

II. Background

A. Committee-based Blockchains

Blockchain is a technology used for maintaining a shared ledger between multiple parties in a decentralized and distributed manner. Blockchain is immutable, transparent, and secure. The security and integrity of blockchains are guaranteed through consensus algorithms. Nodes decide what to append to the blockchain and in which order using consensus algorithms. Among all consensus algorithms, PoW and PoS have been widely used by various blockchains where a single node is selected to create a new block. However, due to the tremendous recourse consumption of PoW and many security problems in PoS, many blockchains have started to adopt BFT protocols as their consensus algorithm [2], [3]. These blockchains, referred to as *committee-based*, often put a committee in charge of producing new blocks instead of a single node. Most committee-based blockchains are leader-based [5], [12], [13]. In these blockchains, the consensus process starts with the round leader proposing a block. Then, other committee members verify the proposed block and vote for it through multiple rounds of communication. The block is committed if a majority of the committee members vote for it.

Tendermint is a committee-based blockchain framework adopted by many blockchain platforms such as Cosmos [14]. In Tendermint, a set of committee members known as *validators* form the committee. In each turn, one of the validators is selected as the proposer after a round-robin scheme. Validators use an algorithm similar to Practical Byzantine Fault Tolerance (PBFT) as the consensus algorithm to vote and decide on the validity of the proposed block. The voting process consists of three steps: pre-vote, pre-commit, and commit. In each step, each process broadcasts its signed message to all other validators. Then, each validator verifies the received messages and moves to the next step if it collects enough messages. If a block gets enough commit messages, it is considered approved, and processes move to the next round.

B. Nash Equilibrium

Nash equilibrium is a concept in game theory that represents a stable state in which none of the players is incentivized to change their strategy. Nash equilibrium analyzes games with rational players trying to maximize their utility. In these games, each player chooses their strategy to better respond to other players' strategies. Nash equilibrium is reached when each player's strategy is the best response considering other players' strategies. In other words, a state is a Nash equilibrium if no player can improve their utility by changing their strategy while other players' strategies are unchanged.

While Nash equilibrium does not guarantee the optimal state, it is still widely used for analyzing the rational player's behavior in different fields. In decentralized setups such as blockchains, it is important that all participants cooperate. Thus having a system in which following the protocol by all players is a Nash equilibrium is desirable. Therefore, many have analyzed the rational behavior in different blockchains, specially committee-based blockchains, and investigated whether following the protocol is a Nash equilibrium [15]–[18]. However, such analyses are not always accessible due to the complexity of such systems and often rely on different assumptions and simplifications.

C. TLC Model Checking

TLA$^+$ is a formal specification language based on the Temporal Logic of Actions (TLA) that allows modeling concurrent systems on a high level [7]. Using TLA$^+$, it is possible to verify and analyze distributed systems through mathematical descriptions of the behavior of their components. TLA$^+$ enables describing the systems through abstraction using variables, state transitions, actions, and temporal properties. It also allows proving either safety properties (something bad never happens) or liveness properties (something good eventually happens). Most TLA$^+$ specifications are in the form of $\mathit{Init} \wedge \Box[\mathit{Next}]_v \wedge L$. *Init* specifies the initial state and initializes the variables. The transition between states is determined by *Next*, with a given tuple of all specification variables v and a liveness property L.

Temporal Logic Checker (TLC) [19] is a model checker that explores all possible reachable states in a TLA$^+$ specification. TLC allows for checking specified properties called *invariant* against the explored system states and determining whether they hold. Each invariant can be a safety condition or a fairness constraint, providing insight into the system's behavior. Also, TLC enables tracing and debugging the specification by generating examples of the violation of the invariants.

III. Methodology

Here we present our method, *Eiffel*, to identify possible Nash equilibria in a given specification using TLA^+. We first discuss the requirements an input specification must fulfill before Eiffel is applied to it. Then we discuss the specification of Eiffel in detail.

A. Input Specification Requirements

Given specifications need to fulfill some requirements to fit in the context of Eiffel. First, as Eiffel is designed to prove Nash equilibrium, an underlying specification needs to have well-defined utility functions for all participants in the system. Eiffel expects the given specification to have a variable for keeping track of each participant's utility (costs and rewards). This variable must be a function that maps each process to its utility.

To prove that a certain set of actions is Nash equilibrium, we need to compare players' utilities when following the protocol and deviating. The deviation from the protocol can be in two different ways. The players can skip some parts of the protocol and not participate or alter some actions. Therefore, Eiffel requires the underlying specifications to separate actions that can be deviated and actions that must be followed. For example, for a rational player to be able to skip one specific step in the algorithm, this step has to be a separate action.

In addition, since Eiffel only verifies the Nash equilibria in the final states, it expects a state condition in the given specification that checks whether the current state is a final state.

B. Eiffel

Here we present Eiffel, a method in TLA^+ that checks whether a certain scenario is a Nash equilibrium in a given specification. In TLA^+ model checking, we can verify *properties* that are either true or false on a single execution. However, to check whether a scenario is a Nash equilibrium, we need to compare the players' utility when following the scenario with the utility of deviating from the scenario. This makes Nash equilibrium a *hyperproperty*, and since it is a predicate on a set of executions, it cannot be verified without self-composition [20]. Therefore, the main idea behind Eiffel is to initialize two instances of the input specification and compare their states.

Figure 1 shows an overview of Eiffel. Eiffel first creates two instances from the input specification. The variables of each instance are assigned with a corresponding variable inside Eiffel. Note that the corresponding variables for the two instances need to be different; therefore, two versions are needed for each variable. On the other hand, having only one version of each constant is enough since both instances use the same constants. In the *Init* predicate, both instances are initialized using their *Init* predicate, and one of the system participants is randomly selected as the rational player.

In the *Next* predicate, we define the transition of both instances. Since we want to compare the corresponding variables of the two instances in similar states, each action in the *Next* predicate should transit both instances to the same state. However, transiting to the same state does not mean following the same strategy. While all processes in the first instance completely follow the protocol, in the second instance, one of the processes deviates. For example, if the processes are supposed to broadcast a certain message to transit from state A to state B, one of the processes in the second instance may not broadcast any message, but it still transits to state B. This further allows to have a comparison between corresponding states having different strategies.

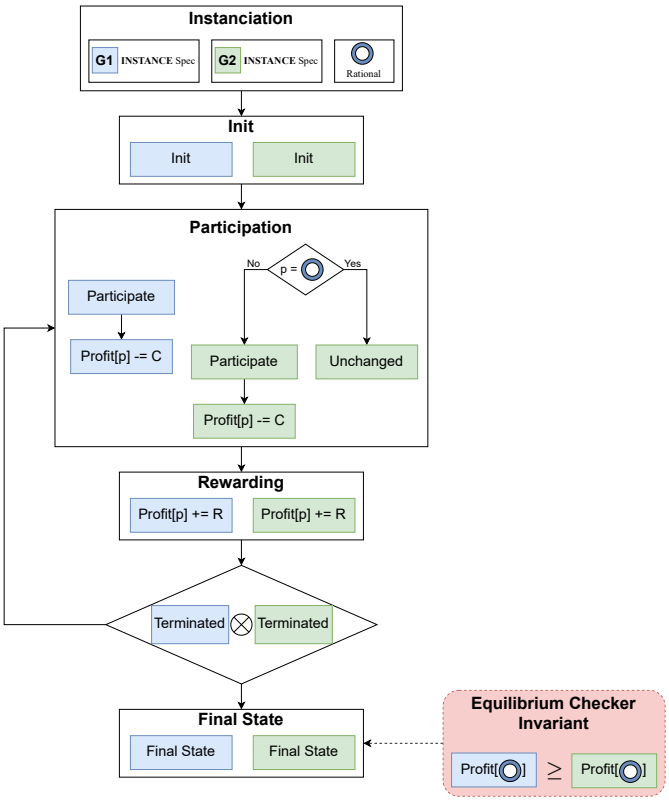

Fig. 1. An overview of Eiffel. Eiffel creates two instances of the given specification and randomly selects one of the players to be rational. It then allows the rational player to skip some parts of the protocol in one of the instances while following the protocol completely in the other one. When the protocol reaches the final state, an invariant for verifying the equilibrium is checked.

When both instances reach a final state, an *invariant* is checked to identify possible equilibria. The invariant compares the profits gained by the processes in both instances, and it ensures that the profit gained by the rational process while behaving is more or equal to when it deviates. If the invariant is violated, it shows that it is possible to gain more by misbehaving, proving that the current strategy is not an equilibrium in the system.

In the next section, we look at a simple example to better show how the methodology works.

IV. Committee-based Blockchains Analysis

Here we present a simple specification of committee-based blockchains in TLA^+, and then we analyze underlying rewarding mechanisms through Eiffel. As Figure 2 illustrates, we follow a simple *propose-vote-commit* procedure presented in [17]. In this abstraction, first, a leader is selected among the processes and proposes a block. Then, other processes validate the block and vote for it. If enough votes are gathered for a block, it is committed. We consider a cost when processes vote for a block and a block reward when a block is committed.

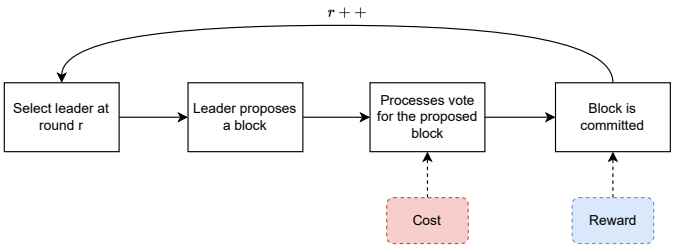

Fig. 2. The abstraction of the committee-based blockchains.

The rewarding mechanism determines which processes receive the reward. Note that both costs and rewards are considered constant, i.e., the same reward is always given to the processes despite the rewarding mechanism.

A. Specification detail

1) variables and constants: There are two constants in the specification. A constant set *Processes* is defined as the list of processes ($\{P_1, P_2, ..., P_n\}$) in the system. Each process proposes one of the blocks in the constant set *Blocks* ($\{B_1, B_2, ..., B_n\}$) when it is a leader. Other variables defined in the specification are as follows:

(a) **Blockchain**: *Blockchain* is a variable for keeping track of the blocks in the blockchain. This variable is a function from each block to a list of properties such as the parent block, the round of the block, the set of processes who voted for this block, whether the block is committed, and whether the block is decided. The difference between committed and decided is that a block may be decided by the processes (all the processes made their decision about whether to vote or not vote for the block), while it is not committed due to needing more votes.

(b) **Decided**: This variable determines whether a process has received and validated the block. Decided is a function from the set of processes to a boolean value. Note that *Decided* is $TRUE$ for a process that validated the new block but has not voted for it yet.

(c) **Utility**: This variable is a function from processes to an integer indicating the corresponding profit. Note that the utility of a process decreases when it endures costs such as voting and increases when it gains block reward.

(d) **NewBlock**: This variable holds the block proposed in the current round. It changes from one round to the other.

2) System states and actions: In the initial state, one of the blocks is randomly selected as the genesis and *NewBlock* is set to it. The parent of all blocks is set to themselves, and their round is set to -1, except for the genesis block, whose round is 0. The value of *Decided* is considered $TRUE$, and the values of *Utility* are set to 0 for all processes.

The procedure begins with selecting one of the processes as the leader. The leader proposes one of the remaining blocks (a block with a round equal to -1). The value of *Blockchain* for the proposed block is changed to the last committed block. Then, as long as *NewBlock* is not decided (at least one of the processes has not still validated it), and it is not yet committed, one of the processes that have not yet voted for this block is selected to validate the block and vote. This action changes the value of *Decided* for the process to $TRUE$. The *Utility* of the process is reduced if it votes. At last, if *NewBlock* has enough votes, it is committed. The processes *Utility* are then updated based on the rewarding mechanism.

TABLE I
RESULTS OF ANALYZING DIFFERENT REWARDING SCHEMES IN COMMITTEE-BASED BLOCKCHAINS USING EIFFEL.

Strategy / Rewarding	Everyone votes		No one votes	
	Equilibrium	**Invariant**	**Equilibrium**	**Invariant**
Reward everyone	No	Violated	Yes	-
Reward only the leader	No	Violated	Yes	-
Reward the voters	Yes	-	Yes	-

B. Equilibrium Analysis

In committee-based blockchains, the protocol's security is guaranteed through the active participation of all committee members in block verification and voting. If the protocol allows for free riding, rational processes skip some parts of the protocol, which eventually makes the system less secure. Therefore, it is important to make sure the reward mechanism discourages free riding. This can be analyzed by checking whether everyone participating in the system is a Nash equilibrium.

Game theoretical analysis shows that in committee-based blockchains, participation of all processes is a Nash equilibrium if we reward only those who have voted for the committed block [17]. However, if we reward all processes or only the leader, participation of all processes is no longer a Nash equilibrium. Figure 3 shows all possible outcomes in a small example with only three processes under different reward mechanisms. As the example shows, if we want to check whether a certain strategy is a Nash equilibrium, we need to compare it with possible states in which only one of the players deviates. For example, for checking state a in Figure 3, we need to compare it with states b, c, and e. The same comparison happens in Eiffel by creating two different instances of the initial specification. The first instance results in the desirable state (state a), while the other instance randomly picks one player to deviate, resulting in one of the other possible states (states b, c, and e).

Table I shows the results of analyzing two different strategies (everyone votes and no one votes) under the three mentioned rewarding mechanisms using Eiffel. Note that the invariant is not violated when a strategy is a Nash equilibrium. Eiffel, therefore, confirms the previous analysis that only rewarding the active voters results in everyone voting to be a Nash equilibrium [17]. Consequently, we can use Eiffel as an extra tool to confirm whether a specific strategy results in a Nash equilibrium.

V. TENDERMINT ANALYSIS

This section uses a one-shot consensus Tendermint specification [11] including all algorithmic details to show how Eiffel

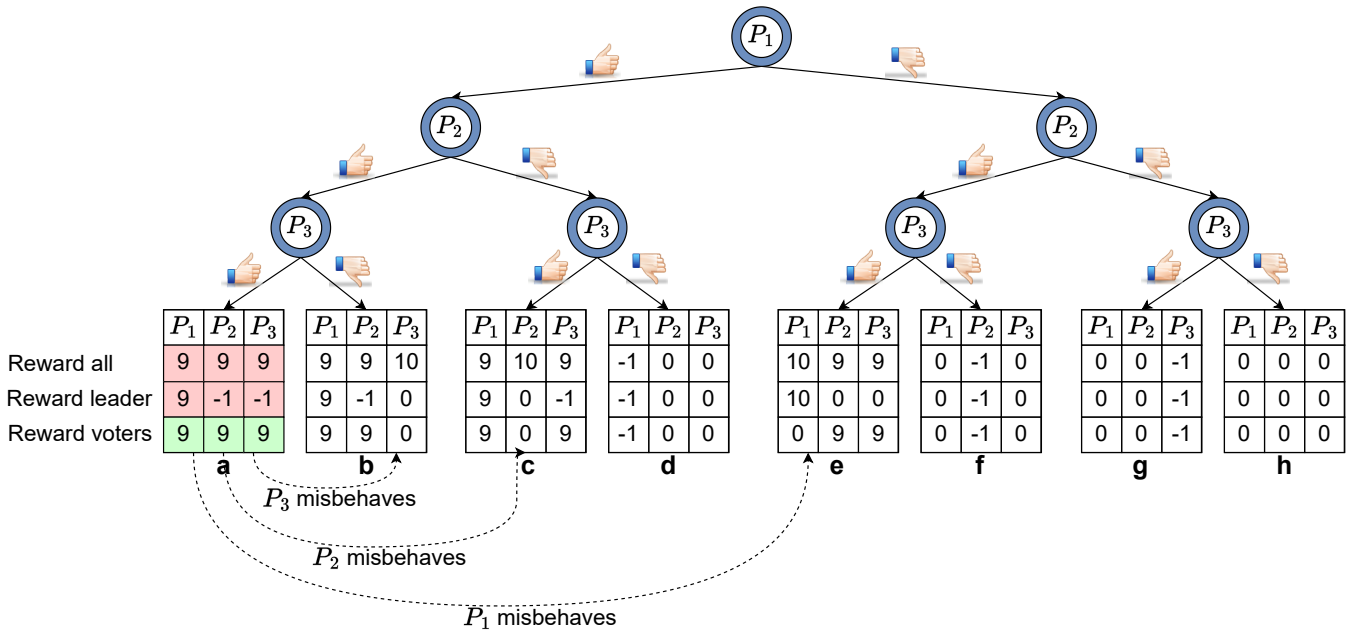

Fig. 3. All possible final states in the committee-based blockchain specification having three processes.

can be used with a fully detailed specification created for a different purpose. The specification follows the pseudo-code in [21]. The detailed consensus protocol is the main difference between this specification and the simplified specification in Section IV. The consensus protocol has four steps: proposal, pre-vote, pre-commit, and decide. In each step, the processes broadcast their messages to all other processes in the system.

Since the original specification does not include rewards, we made some modifications to keep track of the process utilities. We add an extra variable *Utility* to the specification. We consider a cost to broadcasting messages, and the processes' utilities are reduced once they participate in any of the consensus steps. We also add an extra state for rewarding the processes. This is the very last state in the specification, and the processes get rewarded based on the reward mechanism.

We also separate the actions that can be skipped from other actions. In this way, while correct processes always follow the protocol no matter the cost, a rational process can skip costly actions such as broadcasting messages.

We use Eiffel to check whether everyone following the protocol is a Nash equilibrium in Tendermint. We run the specification with three processes where at least two votes are needed in each phase. Broadcasting a message to the network costs $C = 1$ each time, and a constant reward $R = 10$ is given out when a block is approved. We first reward all processes evenly despite their contribution. When running Eiffel, the equilibrium invariant is violated, meaning everyone following the protocol is not a Nash equilibrium. For example, one possible result for the variable $Utility$ when every process follows the protocol is $\langle 8, 8, 8\rangle$ for P_1, P_2, and P_3 respectively. When P_2 is selected as the rational process and skips the consensus steps, the final profits are $\langle 8, 10, 8\rangle$. This shows that

TABLE II
THE NUMBER OF DISTINCT STATES FOR THE TENDERMINT SPECIFICATION AND EIFFEL USING THE SAME SPECIFICATION WITH THE DIFFERENT NUMBER OF PROCESSES.

Method	Distinct States		
	3 processes	4 processes	5 processes
Tendermint specification	10,714	47,778	890,147
Eiffel using Tendermint specification	31,352	194,627	7,413,895

Tendermint suffers from free riding, where processes may get more profit by skipping some rounds of the protocol. One possible solution to the free riding problem is to reward only processes who participated in the consensus. We model this by tracking the processes that participated in all consensus steps and rewarding only those. The equilibrium invariant is not violated when we run Eiffel using this rewarding scheme, proving that everyone participating here is a Nash equilibrium.

To measure the scalability of Eiffel, we run the specification with and without Eiffel. The results are shown in Table II. We see that Eiffel takes longer to run due to running two different instances of the specification. This blow-up increases with the number of processes. Therefore, extending this analysis to very complex specifications may require other techniques.

VI. COSMOS ANALYSIS

In this section, we apply Eiffel on a reward scheme designed for the Tendermint algorithm. We show that Eiffel can be used to analyze attacks and parameter settings. We use Cosmos [22] rewarding scheme as a use case since it is a well-developed blockchain platform using Tendermint as its consensus algorithm.

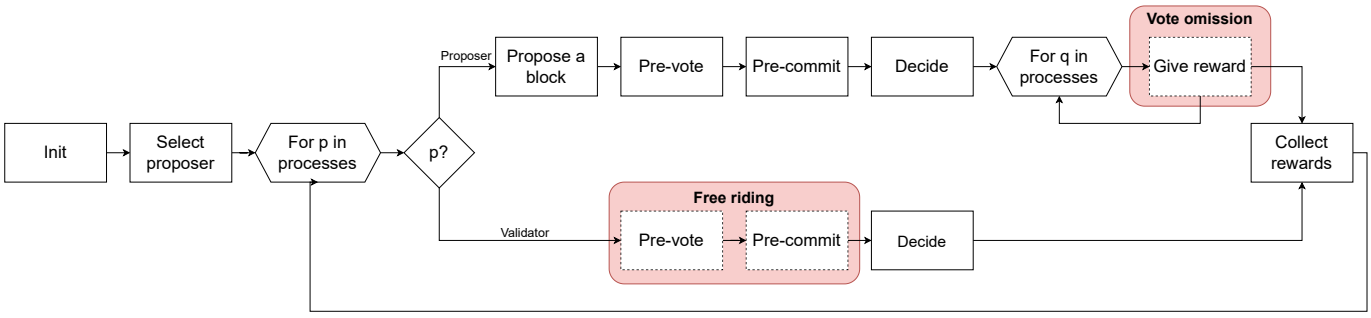

Fig. 4. The abstraction used for modeling rational behaviour in Cosmos. The red boxes indicated the states that a rational process might skip.

In most blockchain platforms, transaction fees constitute an important part of block rewards. Since the transaction fees vary considerably from block to block, giving out a constant reward to every process is not always possible. Therefore, many current schemes, such as Cosmos, divide the rewards equally between all active processes. These schemes normally put leaders responsible for detecting active members through the messages they receive. We used the abstraction illustrated in Figure 4 to model Cosmos rewarding scheme. We first modified the rewarding procedure in the Tendermint specification to divide the rewards between all active processes. Also, we put the leader in charge of deciding whom to reward. As shown in Figure 4, the preliminary Cosmos specification allows for misbehavior based on the process role. While a committee member is able to skip some rounds of the underlying consensus protocol, leaders can skip rewarding some of the processes. Thus in order to have a Nash equilibrium, we should have an equilibrium for both the leader and other processes.

When running Eiffel using this preliminary Cosmos specification, the equilibrium invariant is violated when the rational process is the round leader. The results show that a rational leader gets a larger fraction of the reward by omitting others. Therefore, while using this rewarding scheme, Cosmos no longer suffers from free riding; it is prone to another attack called *vote omission* attack. In this attack, the leader deliberately ignores some of the messages to increase its own share of the rewards [23]. Therefore, using this rewarding scheme, while everyone participating is a Nash equilibrium, the leader acting correctly and rewarding everyone is not a Nash equilibrium.

Cosmos addresses vote omission attacks by giving an additional bonus to the leaders. It is designed in a way to incentivize the inclusion of more processes. The bonus is linear based on the number of committee members the leader decides to reward and is calculated as follows:

$$bonus = \left(b_c + \left(\frac{v-t}{n-t}\right) b_v\right) R \qquad (1)$$

Where v is the number of processes the leader rewards, n is the committee size, t is the minimum number of messages needed for the block to be committed, R is the total reward, and b_c and b_v are the fractions of the reward considered as the constant and variable bonus respectively. At the time of writing the paper, b_c and b_v are equal to 1% and 4% respectively, resulting in the bonus to range between 1% and 5% of the total reward [22].

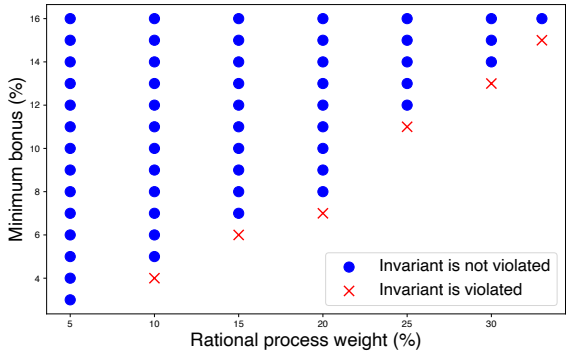

Fig. 5. The minimum bonus where the invariant is not violated considering different weights for the rational process.

The bonus should be large enough to motivate a rational leader to reward as many processes as possible. Baloochestani et al. [23] show that the minimum size of the bonus is directly related to the voting power of the leader. To verify this, we add a constant *Weight* to the specification that indicates each process's voting power. Then we check different values for the bonus (starting with 1%) for different weights to see the minimum bonus required to motivate rational processes to follow the protocol. Then we run Eiffel with different values for the bonus and leader voting power. Figure 5 illustrates the minimum bonus required for the invariant to hold. We see that the minimum bonus is also increased when the attacking fraction increases. This further verifies that the current bonus in Cosmos is sufficient to thwart rational processes with a weight up to 10%.

VII. Conclusion

The security and effectiveness of distributed systems, such as blockchains, heavily rely on the participation of participants. Rewarding mechanisms play a crucial role in incentivizing this participation. However, for these mechanisms to be effective, they need to be fair, correct, and capable

of withstanding rational behavior. Utility analysis, therefore, becomes essential in ensuring the desired outcomes.

Formal verification methods, such as TLA$^+$, have been widely used to ensure the correctness of distributed systems. However, their application for utility analysis has been limited. In this paper, we proposed Eiffel, a novel approach that extends formal verification capabilities in distributed systems to include utility analysis. By leveraging TLA$^+$ and the concept of Nash equilibrium, Eiffel enables the analysis of rewarding mechanisms and the detection of potential attacks. We showed the capability of Eiffel to prove Nash equilibria and to find different attacks in the rewarding mechanisms using committee-based blockchains and Tendermint as the use cases. This paper introduced a new way for utility analysis using TLA$^+$. In the future, Eiffel can be extended to use other formal verification techniques and analyze different types of blockchains.

REFERENCES

[1] S. V. Akram, P. K. Malik, R. Singh, G. Anita, and S. Tanwar, "Adoption of blockchain technology in various realms: Opportunities and challenges," *Security and Privacy*, vol. 3, no. 5, p. e109, 2020.

[2] Y. Meng, Z. Cao, and D. Qu, "A committee-based byzantine consensus protocol for blockchain," in *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 2018, pp. 1–6.

[3] W. Li, S. Andreina, J.-M. Bohli, and G. Karame, "Securing proof-of-stake blockchain protocols," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2017, pp. 297–315.

[4] L. LAMPORT, R. SHOSTAK, and M. PEASE, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 1982.

[5] J. Kwon, "Tendermint: Consensus without mining," *Draft v. 0.6, fall*, vol. 1, no. 11, 2014.

[6] Y. Amoussou-Guenou, A. del Pozzo, M. Potop-Butucaru, and S. Tucci-Piergiovanni, "On fairness in committee-based blockchains," in *2nd International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2020)*, 2020.

[7] L. Lamport, "The temporal logic of actions," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 16, no. 3, pp. 872–923, 1994.

[8] ——, "Byzantizing paxos by refinement," in *International symposium on distributed computing*. Springer, 2011, pp. 211–224.

[9] L. Jehl, "Formal verification of hotstuff," in *International Conference on Formal Techniques for Distributed Objects, Components, and Systems*. Springer, 2021, pp. 197–204.

[10] S. Braithwaite, E. Buchman, I. Konnov, Z. Milosevic, I. Stoilkovska, J. Widder, and A. Zamfir, "Formal specification and model checking of the tendermint blockchain synchronization protocol (short paper)," in *2nd Workshop on Formal Methods for Blockchains (FMBC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[11] Z. Milosevic and I. Konnov, "Tendermintacc_004_draft.tla," https://github.com/cometbft/cometbft/blob/main/spec/light-client/accountability/TendermintAcc_004_draft.tla, 2020, accessed on May 15, 2023.

[12] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, and A. Sonnino, "State machine replication in the libra blockchain," *The Libra Assn., Tech. Rep*, 2019.

[13] J. Chen and S. Micali, "Algorand: A secure and efficient distributed ledger," *Theoretical Computer Science*, vol. 777, pp. 155–183, 2019.

[14] J. Kwon and E. Buchman, "Cosmos: A network of distributed ledgers," *URL https://cosmos. network/whitepaper*, 2016.

[15] W. Li, M. Cao, Y. Wang, C. Tang, and F. Lin, "Mining pool game model and nash equilibrium analysis for pow-based blockchain networks," *IEEE Access*, vol. 8, pp. 101 049–101 060, 2020.

[16] A. Kiayias, E. Koutsoupias, M. Kyropoulou, and Y. Tselekounis, "Blockchain mining games," in *Proceedings of the 2016 ACM Conference on Economics and Computation*, 2016, pp. 365–382.

[17] Y. Amoussou-Guenou, B. Biais, M. Potop-Butucaru, and S. Tucci-Piergiovanni, "Rational behavior in committee-based blockchains," *Cryptology ePrint Archive*, 2020.

[18] A. Benhaim, "Study of nash equilibria in blockchain voting systems," Ph.D. dissertation, University of Pennsylvania, 2022.

[19] Y. Yu, P. Manolios, and L. Lamport, "Model checking tla+ specifications," in *Correct Hardware Design and Verification Methods: 10th IFIP WG10. 5 Advanced Research Working Conference, CHARME'99 Bad Herrenalb, Germany, September 27–29, 1999 Proceedings 10*. Springer, 1999, pp. 54–66.

[20] L. Lamport and F. B. Schneider, "Verifying hyperproperties with TLA," in *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*. IEEE, 2021, pp. 1–16.

[21] E. Buchman, J. Kwon, and Z. Milosevic, "The latest gossip on bft consensus," *arXiv preprint arXiv:1807.04938*, 2018.

[22] C. Hub, "validator faq." [Online]. Available: https://hub.cosmos.network/main/validators/validator-faq.html

[23] A. Baloochestani, L. Jehl, and H. Meling, "Rebop: Reputation-based incentives in committee-based blockchains," in *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 2022, pp. 37–54.