Formal Modeling and Verification of Justification and Finalization of Checkpoints in Ethereum 2.0 Beacon Chain

2024 IEEE 1st Karachi Section Humanitarian Technology Conference (KHI-HTC) | 979-8-3503-7304-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/KHI-HTC60760.2024.10481930

Muhammad Rashid
Dept. of Computer Science
Comsats University Islamabad
Sahiwal Campus, Pakistan
rashid674r@gmail.com

Imran Rasool
Dept. of Computer Science
Comsats University Islamabad
Sahiwal Campus, Pakistan
imranrasoolch@gmail.com

Nazir Ahmad Zafar
Dept. of Computer Science
Comsats University Islamabad
Sahiwal Campus, Pakistan
nazafar@gmail.com

Hamra Afzaal
Dept. of Computer Science
Information Technology University
Lahore, Pakistan
hamraafzaal@hotmail.com

Abstract— **New to the Ethereum platform with version 2.0 is the Beacon chain. Validator status, attestation information, and many more are maintained via the proof-of-stake (PoS) consensus protocol, which is relied upon. The Ethereum 2.0 beacon chain relies on the validation and completion of checkpoints to validate and finish all the blocks associated with those checkpoints. By formally verifying it using the SPIN model checker, this research tackles the issue of the dependability and security of the Beacon Chain's justification and finalization operations. Due of its novelty (launched in 2020), there is little any literature on the subject. Additionally, no previous study has formally verified the beacon chain using the SPIN model checker. The study makes use of PROMELA, a formal specification language, to formally outline the reasoning and finalization method of the Ethereum 2.0 Beacon Chain. Utilizing the SPIN Model Checker, a program graph is generated for this procedure, which formulaically expresses safety features via the use of linear temporal logic (LTL). To make sure everything is in order, we run the SPIN model checker with the program graph and LTL formulae as inputs to see whether the program graph satisfies the properties. This is the formal verification process.**

Keywords— Model Checking, Formal Methods, SPIN, LTL, Blockchain, Beacon Chain, Ethereum 2.0, Program Graph

I. INTRODUCTION

Many companies are launching initiatives in response to the enormous interest in blockchain technology and distributed ledgers. Regardless, one of the most important target markets for the blockchain concept is the banking sector. Among the many uses for blockchain technology are cryptocurrencies like Bitcoin and Ethereum, among others. Ethereum must be understood [1] before anything else. In an effort to build the aggregated technology that may underpin all proposals for exchange-based state machines, the Ethereum project is underway [2]. A major upgrade to the Ethereum platform, Ethereum 2.0 introduces the Reference Chain Convention, a new confirmation-of-stake mechanism, with the primary goal of enhancing Ethereum's efficiency, adaptability, and security. Participants in the convention, known as validators, stake some of the platform's primary currency, Ethereum, in exchange for the opportunity to serve on boards that recommend and approve new blocks [3] Among the many new features coming to the Ethereum 2.0 Beacon Chain in 2020 are the ability to store validator status information and vote data in blocks. The Beacon Chain also keeps track of information about slashing and rewards.

Using the Dafny language, formal verification of the Beacon chain was carried out in [7]. Unfortunately, this verification fails to check for LTL-expressible safety features in the Beacon Chain activities, particularly during the justification and finalization phases. By using the SPIN model checker to validate LTL-expressible safety features, this work offers a remedy to this issue. The primary goals of this study are 1). Prioritize security 2). Understand how crucial SPIN is 3). Strengthen Ethereum's protocol trustworthiness.

In order to ensure that the checkpoints have been properly justified and finalized, the SPIN model checker is used. To develop a program graph, a finite state model, the process is stated using PROMELA, a formal specification language. When it comes to syntax, PROMELA is quite similar to C. While both automatons and program graphs have similarities, program graphs vary from finite automatons. The program graph does not need the end state to be present. After we created the program graph, our first phase of verification was complete. Following this, LTL is used to codify safety characteristics. Properties of liveness and safety may be defined with its help. In our possession so far are the program graph, an automated model of the justification and finalization process, and the LTL formula for the qualities that describe the system method. The model checker checks the LTL formulae on the program graph after receiving the program graph and the LTL formulas as inputs. When running the SPIN model via the Ethereum 2.0 Beacon Chain, the results of the verification of the justification and finalization process are shown.

Here is the breakdown of the remaining sections of the paper: In Section II, relevant research is provided. Section III provides context on Ethereum and model verification. Part IV provides a concise explanation, accompanied by a flowchart and algorithm, of the processes involved in finalization and justification. It is formalized to specify and verify. The finalization technique and evidence that the reasoning is accurate are presented in Section V. In Section VI, we wrap up all of the verification and results work.

II. Related Work

Give an account of the activities that were part of this study here. With the help of the Coq verification tool, the Gasper convention of Ethereum 2.0 is officially verified in [3]. Mixing Casper and Ghost, Gasper is a new breed. You may vote for the Beacon Chain leader via Ghost, and Casper is a device that helps with finalization. For Phase 0 of the Ethereum 2.0 Beacon Chain, an executable K-model is created in [4]. This study presents the model with a high degree of abstraction. One of the foundations of the Gasper protocol is the Casper protocol, which is formally verified in [5]. We have completed the confirmation using the Coq proof assistance. Casper represents the shift in Ethereum 2.0 from the Proof of Work (PoW) standard to the Proof of Stake (PoS) standard. In [6], the Correct-by-Construction Casper convention is properly verified with the help of Coq proof aid. Another option for Casper conventions is Correct-by-Construction. It differs somewhat from the Casper FFG's (finalization-friendly gadget). For the purpose of formal verification, the Beacon Chain is deductively verified in [7]. To carry out this formal verification, the Dafny language is used. The Ethereum 2.0 creator himself announced Casper and outlined all of its features in [8]. Verilay was the first chain relay system that allow validating PoS protocols to generate finished blocks [9]. This included projects like Polkadot, Ethereum 2.0, and Cosmos. None of the original blockchain protocols or validator processes need to be altered to implement the idea. Validation of block proposer signatures is carried out via a specific relay smart contract on the target blockchain. Authors in [10] use 30% of the entire stake to conduct cost and probability evaluations of Beacon Chain attackers. An exhaustive overview of the steps taken to get Ethereum up to version 2.0, or the switch from the PoW to the PoS protocol, can be found in [11]. To understand the level of development of Proof-stake-BFT concepts, as well as to identify outstanding questions and continuous research difficulties, [12] presents the key aspects and distinctions of these proposals in relation to Bitcoin. Concerns with Ethereum smart contracts are resolved in [13]. This research provides a thorough examination of the stated issues. In [14], the use of blockchain technology is examined, and a new authentication service, DAuth, built on the Ethereum Blockchain, is suggested. There is also a prototype that allows users to authenticate on the site. Using the Ethereum blockchain, a system is developed in [15] to handle home applications. This paper proposes an architecture based on Web 3.0 that does away with centralized agencies and advocates for an internet that is completely decentralized, safe, and transparent [16]. With Ethereum and smart contracts, a safe, decentralized method of starting data-based payments may be created. An examination of blockchain-based smart contracts and their associated security flaws is discussed in [17]. The research has offered a strategy to decentralize the e-commerce platform leveraging blockchain technology, smart contracts, and decentralized storage systems like IPFS (Inter Planetary File System) in [18]. For IEC 61499 applications, the FBME-modular IDE's debugging activities may be unified, according to [19]. Plus, they demonstrate the model's operation on a basic example and provide the formal model of the execution trace format. To make it straightforward to implement in the SPIN model checker, the authors address the translation of the UML diagram into the PROMELA language in [20]. The timed automaton is described and verified with the help of the SPIN model checker in [21]. The flying software in [22] is checked using an SPIN model checker. The document addresses the formal aspects of the spacecraft's operation. As stated in [23], the communication protocols and rules are verified formally using the SPIN model checker for the distance vector routing protocol. The authors of [24] discuss smart waste management systems, where they include several methods to model such systems with the use of the Internet of Things (IoT) and blockchain technology. For the purpose of automating and modeling waste management systems, the integrated technologies include blockchain, the Internet of Things (IoT), UML, and TLA+. In [25-27], you may find further work on formal modeling using VDMSL.

This research is the first of its kind to investigate formal verification for the Beacon Chain inside the SPIN framework, and it uses the SPIN model checker to conduct an innovative analysis, specification, and verification of Ethereum 2.0. The uniqueness and importance of our attempt in improving the security and dependability of Ethereum 2.0 are emphasized by our major focus on the rigorous formal verification of the justification and finalization procedures.

III. Background

The information on Ethereum 2.0, Beacon Chain, and Model Checking is presented in this part about the background.

A. Ethereum

By far, the most sophisticated Blockchain-based financial application is Ethereum. In 2015, Vitalik Buterin established Ethereum. The Ethereum Foundation had intended to switch to the Proof of Stake consensus mechanism from the older Proof of Work protocol, which Ethereum was using at the time [2].

B. Ethereum 2.0

With the release of Ethereum 2.0 in 2018, the network's consensus mechanism changed from PoW to PoS. With its three improvements, Ethereum 2.0 achieves its three main goals: scalability, decentralization, and security. First, Beacon Chain; second, Sharding; and third, changes. Computational state [3].

C. Beacon Chain

The Beacon Chain kept information on the validator's attestations, rewards, and penalties and maintained the validator's lifecycle. Some major terminologies in the Beacon Chain used in paper are given below.

Proof of Stake (PoS): Consensus protocol in which participants stake 32 ethers to become active validators.

Validator: These are active participants in Ethereum which validate the blocks.

Attestation: The block proposed is validated by the votes of 128 validators which is attestation.

Committee: Set of 128 validators that perform attestation of proposed blocks.

Checkpoint: Each epoch boundary like 32th slot is checkpoint of that epoch
Epoch: Set of 32 slots with timespan of 6.4 minutes
Slot: 12 seconds time span in which a block is proposed
Genesis Epoch: Initial epoch in Beacon Chain
Genisis Block: Initial block in epoch
Genisis checkpoint: Initial checkpoint

D. Model Checking

A method for verifying systems that use both software and hardware is known as model checking. To make sure the model is accurate, model checking checks several system attributes according to the system's requirements.

E. SPIN Model Checker

Software tools like the SPIN model checker can verify distributed and concurrent systems. Its stellar reputation originates from its ability to validate models expressed in the PROMELA modeling language using the SPIN model checking technique.

IV. JUSTIFICATION AND FINALIZATION PROCEDURE

In this Section, explain the justification and finalization checkpoints in the Beacon Chain. As in the previous section, describes the important terminologies of the Beacon Chain. In Beacon Chain, after every 32 slots which is equal to one epoch a checkpoint is made. Blocks in that checkpoints are validated by 128 members of the committee which is selected randomly. Checkpoints are validated by all the attestations made by active validators in the committee. Fig 1 is given to understand justification and finalization.

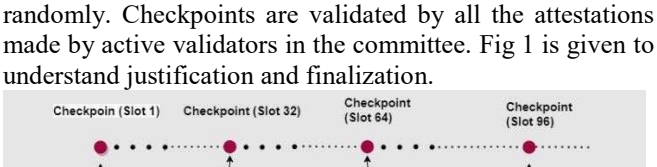

Fig. 1. Overview of Justification&finaliztion

As in fig 3 describes that the genesis checkpoint is already justified and finalized. In case a checkpoint is not genesis then if it gets 2/3 attestation then the checkpoint will be justified and the previous checkpoint which is already justified will be finalized. A flow diagram for complete system understanding is given.

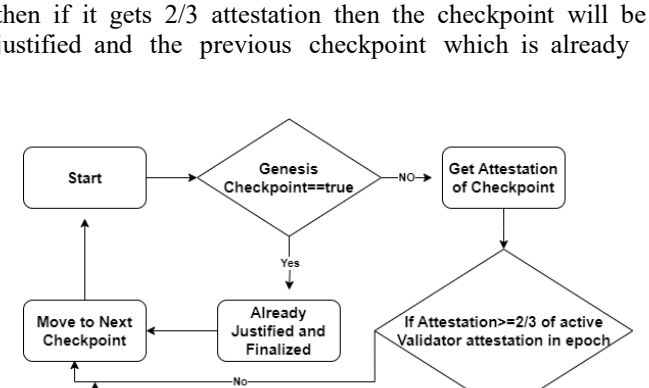

Fig. 2. Flow diagram of Procedure

The subprocesses of justification and finalization are illustrated interacting with each other in fig. 2, along with certain essential requirements in both processes. A mathematical method is provided to comprehend the operation of the whole system.

Algorithm: Justification and Finalization

```
1   Justification&finaliztion (C, JF, J)
2   ∃ c ∈ C
3   if (genesis(c)==true)
4   JF=JF ∪ {c} then return
5   Else if (c ∉ J && genesis(c)≠true)
6     Get_Attestation (c)
7     R←Required_Attestation (Active_validators,
8     attestations, c)
9     if ((Get_Attestation(c)>=R) && (c-1 ∈ J))
10        J=J\{c-1}
11        JF=JF ∪ {c-1}
12        J=j ∪ {c}
13  End
```

An algorithm is given to understand the justification and finalization operation of the Beacon Chain. In the first line, C is a set of Checkpoints, JF is a set of justified and finalized checkpoints, and J represents only justified checkpoints. In line 3, it is checked that if a checkpoint is genesis, then it should be justified and finalized. In case it is not genesis then the attestation of the checkpoint will be checked and will be compared to the required attestation R. If the calculated attestation of the checkpoint is greater or equal to the required attestation, then the previous justified block will become justified and finalized, and the current checkpoint, whose attestation is found, will become justified but not finalized. If all the described conditions are fulfilled in the next checkpoint, then this checkpoint will also become finalized

V. FORMAL MODELING OF JUSTIFICATION AND FINALIZATION

Complete the system's formal specification in the PROMELA language. When using the SPIN model checker, PROMELA is the default language. The C language is quite similar to its syntax. Create the system's program graphs using this language. The purpose of this section is to describe every single piece of code.

A. Program Graph for Justification and Finalization

A program graph serves as an automaton derived from the specification of a system written in PROMELA. Unlike finite automata, where a final state is mandatory, program graphs exhibit flexibility in this regard. They may or may not include a final state, distinguishing them from traditional finite automata. Consequently, a program graph is characterized by a set of five tuples, allowing for variations in its structure and the presence of a final state as dictated by the specific system specification.

- Set of locations: it is like state of automata

- Initial location: The initial state or location in a program graph represents the starting point of program execution and serves as the entry point for the analysis of system behavior.
- Set of conditions: Conditions in a program graph define logical predicates guiding state transitions based on specific criteria within the program's logic.
- Initial condition: beginning state is defined in the program flow diagram at any point.
- Transition function: programmatic arrow labeled with messages and criteria.

Begin by outlining the code specification, organizing it in a sequence starting with data types followed by successive blocks addressing system verification, and then elucidate each step through distinct blocks.

```
mtype={start,compute_current_epoch,attestion_between_previous_or_current_epoch,got_attestation,got_total_active_balance,get_previous_epoch_attesting_balance,get_current_epoch_attesting_balance,previous_epoch_justified,current_epoch_justified,find_current_target_balance
,got_current_target_balance,fourth_as_source,previous_checkpoint_finalizes,current_checkpoint_finalizes,third_as_source ,second_as_source,first_as_source }
int current_epoch_value=7
int genesis_epoch=0
bit get_current_epoch=1
bit get_previous_epoch=1
bit previous_attestation
bit current_attestation
bit get_active_validator_indices
bit get_unslashed_attesting_indices
int effective_balance_increment=2
int active_validators_effective_balance=3
int total_balance=5
bit  got_previous_target_balance
bit previous_justifed_checkpoints
bit  current_justified_checkpoints
bit justification_bits=1
int  previous_epoch_balance=7
int  current_epoch_balance=7
int previous_epoch=5
int current_epoch
int bits=3
```

This block of code describes declarations of variables in our modeling procedure. Major datatypes used are integer, bit and mtype. Integer is used for numeric values and bit represents0 and 1. Mtype is message type datatype which shows symbols and messages only.

1. *Genesis Epoch*

```
Genesis_Epoch:
start
→compute_current_epoch
→ if
:: current_epoch_value <=genesis_epoch+1
→ goto Genesis_Block
:: current_epoch_value > genesis_epoch +1
```

Initially it should be ensured that checkpoint block should not be in genesis epoch. The Genesis epoch is already justified and finalized, so skip the justification and finalization of the Genesis epoch. If the current epoch is not genesis, then move forward with further procedures.

2. *Getting Attestation*

```
Getting_Attestations:
→  attestion_between_previous_or_current_epoch
if
:: (get_current_epoch==0) || (get_previous_epoch==0)
→ goto Getting_Attestations
::  (get_current_epoch==1) && (get_previous_epoch==1)
→ previous_attestation=previous_attestation+1
→ current_attestation=current_attestation +1
→ got_attestation
```

After confirming that the epoch is not a genesis epoch, proceed to obtain attestations from both the current epoch and the previous epoch. Both attestations are pivotal as they significantly contribute to the justification and finalization processes, ensuring a comprehensive evaluation for the system's integrity and accuracy.

3. *Getting Total Balance*

```
Getting_total_balance:
→get_active_validator_indices=get_active_validator_indices+1
→ total_balance=effective_balance_increment+
active_validators_effective_balance

→ got_total_active_balance
```

This block of code shows that after attestation, the total balance of all active validators should be known; therefore, in this block, get the total balance of active validators.

4. *Previous Epoch Balance*

```
Getting_previous_target_balance:
→get_unslashed_attesting_indices=get_unslashed_attesting_indices+1
→  get_previous_epoch_attesting_balance
→got_previous_target_balance=got_previous_target_balance+1
→ find_current_target_balance
```

Previous epoch, which is justified, find the total balance of that epoch. The balance of all the validators involved in the previous epoch will be considered in this block of code.

5. *Current Epoch Balance*

```
Getting_current_target_balance:
→ get_unslashed_attesting_indices
→get_current_epoch_attesting_balance
→got_current_target_balance
```

The current epoch, which we want to justify, will be considered here to calculate the balance of active validators participating in this epoch.

6. Justification & Finalization(bit=0)

```
Justification:
if
:: justification_bits==0->
atomic{previous_epoch_balance*3>=total_balance*2
->
previous_justifed_checkpoints=previous_justifed_checkpoin
ts+1
-> previous_epoch_justified->
}
→ if
:: bits==2
→ second_as_source
→current_epoch_value==previous_epoch+2
→ current_checkpoint_finalizes
:: bits==1
→ first_as_source
→ previous_epoch+1==current_epoch_value
→ current_checkpoint_finalizes
Fi
```

In this block of code, perform the justification and finalization of checkpoints after all the steps mentioned above. The justification bit is kept at 0, and source epochs are considered 2 and 1.

7. Justification and Finalization (bit=1)

```
:: justification_bits==1→
atomic{ current_epoch_balance*3>=total_balance*2
→current_justified_checkpoints=current_justified_checkpoi
nts+1
→ current_epoch_justified}→
→ if
:: bits==4
→fourth_as_source
→ current_epoch=previous_epoch+3
→ previous_checkpoint_finalizes

:: bits==3
→ third_as_source

→ previous_epoch+2==current_epoch_value
→ previous_checkpoint_finalizes
```

In this block of code, perform the justification and finalization of checkpoints after all the steps mentioned above. The justification bit is kept 1, and source epochs are considered 4 and 3. These two blocks above are the main blocks of the code for modeling procedures.

B. LTL Formulas for properties

Programmatic arrow labelled "transition"To define the attributes, one uses linear temporal logic. A formula is formalized by LTL from a description of a property, which may be either safety or liveness. Here, let's formalize two safety features into an LTL formula and use them for the justification and finalization operations. In the calculations, we used symbols that represent situations and messages:

- *[]*: Within these square brackets, the whole route in the program graph is analyzed, together with all of its states.
- <>: This symbol pertains to certain states within a complete path, rather than encompassing all states along the path.
- *X*: This symbol takes into account the state that follows the one where we validate our condition.

a. Safety properties

Safety properties are categorized into negative and positive types, with negative safety properties aiming to prevent undesirable events, such as ensuring "two trains should not occupy the same track." In contrast, positive safety properties focus on promoting desired outcomes, like "after a yellow signal, there should be a green or red signal." These distinctions align with specific requirements, allowing the formulation of safety properties based on the given specifications. The provided LTL formulas and their descriptions further articulate these safety properties.

Property 1:

If Current checkpoint is justified then previous justified checkpoint will be finalized.

LTL formula of this property is given below

```
ltl p1{ [](<>(current_epoch_justified → previous_checkpoint_finalizes))}
```

Property 2:

If current checkpoint gets 2/3 attestation than it will be justified

LTL formula of this property is given below

```
ltlp2{[](validator_effective_balance>=32)→X(<>(enter_in
_queue))→X(<>(validator_activated))}
```

C. Verification And Results Analysis Through Spin

Done verification through SPIN in following steps:

- Create program graph for the system of justification and finalization process.
- Define safety properties for system to ensure its correctness.
- Formalize safety properties into LTL formulas.
- Give LTL formula and program graph as input to the SPIN model checker.
- Model checker after verification will give results whether properties are satisfied or not.

a. Verification results

Results or proof of correctness are given using the SPIN model checker. All the properties are verified using the model checker.

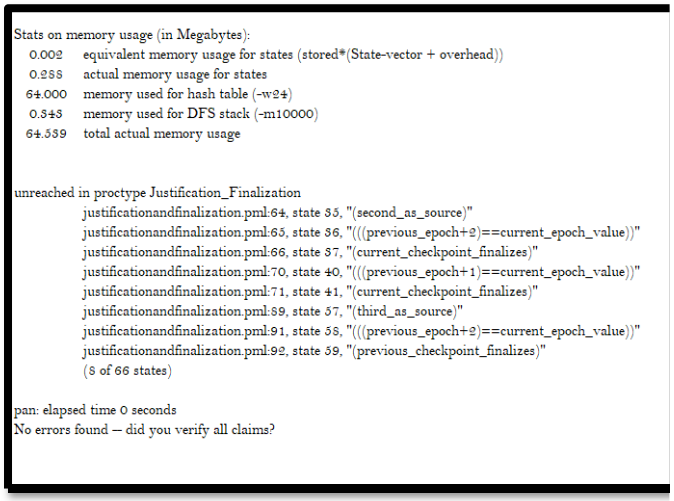

Fig.3. Proof of Correctness

Our program graph for the purpose of justifying and finalization the checkpoint operation satisfies all properties, as shown in the picture, and no errors are detected.

VI. CONCLUSION

In this study, we conducted formal modeling verification of the Ethereum 2.0 Beacon Chain's Justification and Finalization procedure using the SPIN model checker. Initially, we provide a concise overview of the justification and finalization process, followed by its formal specification in the PROMELA language to generate a program graph via the SPIN model checker. The program graph illustrates the entire process. Subsequently, safety and liveness properties are defined using LTL formulas to ensure procedural correctness. The program graph and LTL formulas serve as inputs for the model checker, employing model checking for effective verification and heightened system correctness. Results indicate satisfaction of LTL formulas by our program graphs. This research contributes to Blockchain, particularly in the realm of financial applications, enhancing safety measures. However, it is limited to safety properties verification for the beacon chain. Future work may extend this by verifying liveness properties in the justification and finalization procedure.

ACKNOWLEDGMENT

This project is partially funded by Ethereum Foundation under Grant ID: FY22-0695.

References

[1] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, "Blockchain," *Bus. Inf. Syst. Eng.*, vol. 59, no. 3, pp. 183–187, 2017.

[2] G. Wood, "Ethereum: a secure decentralised generalised transaction ledger," *Ethereum Proj. yellow Pap.*, vol. 151, pp. 1–32, 2014.

[3] M. A. Alturki, E. Li, D. Park, and B. Moore, "Verifying Gasper with Dynamic Validator Sets in Coq," 2020, [Online]. Available: https://github.com/runtimeverification/beacon-chain-verification/blob/master/casper/report/report.pdf

[4] M. A. Alturki, D. Bogdanas, C. Hathhorn, and D. Park, "An Executable K Model of Ethereum 2 . 0 Beacon Chain Phase 0 Specification".

[5] K. Palmskog, M. Gligoric, L. Pena, B. Moore, and G. Rosu, "Verification of Casper in the Coq Proof Assistant," 2018, [Online].Available:https://github.com/runtimeverification/casper-proofs/blob/master/report/report.pdf

[6] E. Li, T. Serbanuta, D. Diaconescu, V. Zamfir, and G. Rosu, "Formalizing Correct-by-Construction Casper in Coq," *IEEE Int. Conf. Blockchain Cryptocurrency, ICBC 2020*, pp. 26–28, 2020.

[7] F. Cassez, J. Fuller, and A. Asgaonkar, *Formal Verification of the Ethereum 2.0 Beacon Chain*, vol. 13243 LNCS. Springer International Publishing, 2022.

[8] V. Buterin and V. Griffith, "Casper the Friendly Finality Gadget," pp.1–10,2017,[Online].Available: http://arxiv.org/abs/1710.09437

[9] M. Westerkamp and M. Diez, "Verilay: A Verifiable Proof of Stake Chain Relay," *IEEE Int. Conf. Blockchain Cryptocurrency, ICBC 2022*, pp. 1–9, 2022.

[10] M. Neuder, D. J. Moroz, R. Rao, and D. C. Parkes, "Low-cost attacks on Ethereum 2.0 by sub-1/3 stakeholders," pp. 1–11, 2021, [Online]. Available: http://arxiv.org/abs/2102.02247

[11] D. Ramos and G. Zanko, "Ethereum 2.0–A Review of the Causes and Consequences of the Upcoming Update to Ethereum's Mainnet," *Mobileyourlife.Com*, 2022, [Online]. Available: https://www.mobileyourlife.com/s/ETH_2_SciPaper_2.pdf

[12] S. Tucci-Piergiovanni, "Keynote: Blockchain consensus protocols, from Bitcoin to Ethereum 2.0," *2022 IEEE Int. Conf. Pervasive Comput. Commun. Work. other Affil. Events (PerCom Work.*, pp. 1–1, 2022, doi: 10.1109/percomworkshops53856.2022.9775195.

[13] M. Staderini, C. Palli, and A. Bondavalli, "Classification of Ethereum Vulnerabilities and their Propagations," *2020 2nd Int. Conf. Blockchain Comput. Appl. BCCA 2020*, pp. 44–51, 2020..

[14] S. Patel, A. Sahoo, B. K. Mohanta, S. S. Panda, and D. Jena, "DAuth: A Decentralized Web Authentication System using Ethereum based Blockchain," *Proc. - Int. Conf. Vis. Towar. Emerg. Trends Commun. Networking, ViTECoN 2019*, pp. 1–5, 2019.

[15] Q. Xu, Z. He, Z. Li, and M. Xiao, "Building an Ethereum-Based Decentralized Smart Home System," *Proc. Int. Conf. Parallel Distrib. Syst. - ICPADS*, vol. 2018-Decem, pp. 1004–1009, 2019,.

[16] K. B. Muthe, "DECENTRANET - AN ETHEREUM , PROXY RE-ENCRYPTION," pp. 1–5, 2020.

[17] Z. Jiang, Z. Zheng, K. Chen, X. Luo, X. Tang, and Y. Li, "Exploring Smart Contract Recommendation: Towards Efficient Blockchain Development," *IEEE Trans. Serv. Comput.*, vol. 16, no. 3, pp. 1822–1832, 2023.

[18] B. S. Liya, S. Pritam, S. Rohit Krishna, and K. Navin, "Decentralized E-Commerce Platform Implemented using Smart Contracts," *Proc. - 2023 3rd Int. Conf. Smart Data Intell. ICSMDI 2023*, pp. 23–27, 2023.

[19] T. Liakh, R. Sorokin, D. Akifev, S. Patil, and V. Vyatkin, "Formal model of IEC 61499 execution trace in," *2022 IEEE 20th Int. Conf. Ind. Informatics*, pp. 588–593, 2022,

[20] A. Amirat and A. Menasria, "Automatic Generation of PROMELA Code from Sequence Diagram with Imbricate Combined Fragments," pp. 1–6, 2012.

[21] N. S. Kumar, G.S. Kumar, "Modeling and Verification of Timed Automaton Based Hybrid Systems Using Spin Model Checker," 2016.

[22] P. R. Gliick, B. Labs, and M. Hill, "Using SPIN1 Model Checking for Flight Software," pp. 105–113, 1978.

[23] K. Bhargavan, D. Obradovic, and C. A. Gunter, "Formal Verification of Standards for Distance Vector Routing Protocols," vol. 49, no. 4, pp. 538–576, 2002.

[24] S. Latif, N. A. Zafar "Blockchain and IoT Based Formal Model of Smart Waste Management System Using TLA +," pp. 304–309, 2019, doi: 10.1109/FIT47737.2019.00064.

[25] M. Iqbal, N. A. Zafar, E. H. Alkhammash, "Formally Identifying COVID-19 Patients for Providing Medical Services using Drones," pp. 0–5, 2021.

[26] S. Latif, N. A. Zafar "Intelligent Traffic Monitoring and Guidance System for Smart City," 2018.

[27] A. Rehman and S. Latif, N. A. Zafar "Automata Based Railway Gate Control System at Level Crossing," *2019 Int. Conf. Commun. Technol.*, no. ComTech, pp. 30–35, 2019.