2022

Formal Modeling and Verification of a Blockchain-Based Crowdsourcing Consensus Protocol

Hamra Afzaal

Muhammad Imran

Muhammad Umar Janjua

Sarada Prasad Gochhayat
Old Dominion University, sgochhay@odu.edu

Original Publication Citation

Afzaal, H., Imran, M., Janjua, M. U., & Gochhayat, S. P. (2022). Formal modeling and verification of a blockchain-based crowdsourcing consensus protocol. *IEEE Access, 10*, 8163-8183. https://doi.org/10.1109/ACCESS.2022.3141982

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 24, 2021, accepted December 21, 2021, date of publication January 11, 2022,
date of current version January 21, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3141982

Formal Modeling and Verification of a Blockchain-Based Crowdsourcing Consensus Protocol

HAMRA AFZAAL[1], MUHAMMAD IMRAN[2], (Member, IEEE), MUHAMMAD UMAR JANJUA[1], AND SARADA PRASAD GOCHHAYAT[3]

[1]Department of Computer Science, Information Technology University, Lahore 54000, Pakistan
[2]School of Engineering, Information Technology and Physical Sciences, Federation University, Brisbane, QLD 4000, Australia
[3]Virginia Modeling, Analysis, and Simulation Center, Suffolk, VA 23435, USA

Corresponding author: Hamra Afzaal (hamraafzaal@hotmail.com)

ABSTRACT Crowdsourcing is an effective technique that allows humans to solve complex problems that are hard to accomplish by automated tools. Some significant challenges in crowdsourcing systems include avoiding security attacks, effective trust management, and ensuring the system's correctness. Blockchain is a promising technology that can be efficiently exploited to address security and trust issues. The consensus protocol is a core component of a blockchain network through which all the blockchain peers achieve an agreement about the state of the distributed ledger. Therefore, its security, trustworthiness, and correctness have vital importance. This work proposes a Secure and Trustworthy Blockchain-based Crowdsourcing (STBC) consensus protocol to address these challenges. Model checking is an effective and automatic technique based on formal methods that is utilized to ensure the correctness of STBC consensus protocol. The proposed consensus protocol's formal specification is described using Communicating Sequential Programs (CSP#). Safety, fault tolerance, leader trust, and validators' trust are important properties for a consensus protocol, which are formally specified through Linear Temporal Logic (LTL) to prevent several security attacks, such as blockchain fork, selfish mining, and invalid block insertion. Process Analysis Toolkit (PAT) is utilized for the formal verification of the proposed consensus protocol.

INDEX TERMS Blockchain, consensus protocol, crowdsourcing, model checking, PAT.

I. INTRODUCTION

Crowdsourcing is an effective way to solve complex problems by outsourcing to a crowd of people [1]. In recent years, it has gained considerable attention in academia and adoption in the industry. A large number of companies are using crowdsourcing as a method to solve critical problems. Human intelligence-based crowdsourcing consists of service consumers, service providers, and a crowdsourcing platform, as shown in Figure 1.

Mechanical Turk [2], Upwork [3] and Uber [4] are some of the famous crowdsourcing platforms. In crowdsourcing, service consumers post tasks through a crowdsourcing platform that are hard to solve for computers but are comparatively easy for humans. In some platforms, such as Upwork, a service consumer needs to deposit the payment as an escrow amount to a crowdsourcing platform before a task begins. The interested service providers receive tasks through a crowdsourcing platform. They compete to solve and submit solutions of tasks to the crowdsourcing platform. The service consumers then select the appropriate solutions, and the corresponding service providers get the task payment.

FIGURE 1. Traditional crowdsourcing system.

The associate editor coordinating the review of this manuscript and approving it for publication was Yang Liu.

Crowdsourcing has a wide range of applications in various security-critical systems, for example, disaster management systems [5], traffic monitoring systems [6], and healthcare systems [7]. With the rapid development of crowdsourcing applications, the limitations of traditional crowdsourcing systems are exposed. Firstly, the traditional crowdsourcing systems are based on centralized servers, which can suffer from a single point of failure [8]. Secondly, in case of a dispute between service providers and consumers, issues of free riding and false reporting can occur due to a lack of accountability and effective trust management [9]. In free riding, service providers try to get rewards without providing satisfactory solutions, and in false reporting, service consumers intentionally get useful solutions without losing the deposit. Thirdly, crowdsourcing systems are vulnerable to various security attacks, such as sybil, distributed denial of service (DDoS), and hijacking attacks [10]. Fourthly, sensitive information of users is at risk of privacy disclosure during the process of task assignment [11]. Lastly, there is minimal work on the formal verification of crowdsourcing systems to ensure the correctness, consistency, and completeness of the algorithms and protocols [12]. The advent of blockchain technology [13] brings high hopes to overcome most of the drawbacks in traditional crowdsourcing systems. A consensus protocol is the most critical component of blockchain which can establish mutual trust among crowdsourcing participants in a decentralized way. Therefore, crowdsourcing issues are solved using blockchain consensus algorithms.

A. MOTIVATION FOR A NEW CONSENSUS PROTOCOL

Most of the existing generic blockchain consensus protocols have certain limitations. For example, Proof of Work (PoW) is highly resource-intensive and difficult to apply for large-scale online systems [13]. Employing Proof of Stake (PoS) in crowdsourcing systems may compromise the rights of nodes [14] as it suffers from security, fairness, and centralization issues. Delegated Proof of Stake (DPoS) handles the fairness issue by democratic voting, but still, it has some degree of centralization [15]. Moreover, employing DPoS cannot avoid collusion among stakeholders. Therefore, an application-specific blockchain consensus protocol is highly desirable, considering the peculiarities of crowdsourcing systems [16], [17]. To address the security and trust challenges, some recent studies [18]–[20] have proposed blockchain-based consensus protocols for crowdsourcing applications that also strive to handle accountability, credible crowdsourcing, free riding, and false reporting, and compelling trust management issues. These studies were the prime motivation behind this work.

Several drawbacks are identified in the existing blockchain-based crowdsourcing consensus protocols. For example, Proof of Trust (PoT) elects a leader from ledger nodes using Reliable-Replicated-Redundant And Fault-Tolerant (RAFT) and validators from crowdsourcing service participants based on trust values [18]. However, PoT analyzes some of the security attacks, including sybil, DDoS, and collusion through simulations, and theoretically describes a few of the security properties. An improved PoT is a reputation-based protocol in which a high reputation worker is selected as a miner of the block, and verification nodes are also selected based on reputation [19]. However, the reputation calculation is based on the historical interaction of a miner and feedback of other nodes that may not be reliable. It is based on simulation techniques and analyzes collusion attacks. Mobile Crowdsourcing (MCS) chain is based on the total payment for block generation which informally analyzes security properties [20]. The consensus protocol of zkCrowd is based on DPoS and Practical Byzantine Fault Tolerance (PBFT) and is validated using simulation techniques, which favors voters having more tokens [21]. The RAFT and PBFT used in the existing blockchain-based crow-dsourcing consensus protocols are formally verified [22]–[24], therefore, PoT and zkCrowd are partially verified. The existing blockchain-based crowdsourcing consensus protocols discussed-above employ simulation techniques that do not assure the models' correctness as simulations are performed on limited data set.

B. CHALLENGES

The weaknesses of the existing blockchain-based crowdsourcing consensus protocols present the following challenges.

1) How to design a secure and trustworthy consensus protocol that applies to crowdsourcing services?
   a) How to prevent security attacks, such as blockchain fork, invalid block insertion, and selfish mining?
   b) How to select trusted leader and validators to prevent their malicious behaviors?
2) How to ensure the correctness of the blockchain-based crowdsourcing consensus protocol?

C. CONTRIBUTIONS

This research work aims to present a formally verified Secure and Trustworthy Blockchain-based Cr-owdsourcing (STBC) consensus protocol to deal with the above-discussed challenges. The consensus protocol is nam-ed as STBC because we have defined the security properties of the protocol and trustworthy selection process of a leader and validators based on trust factors. The peculiarities of the STBC consensus protocol are:

1) It records the activities of service providers and consumers, e.g., post, receive, and perform tasks as transactions, and the transactions that gain maximum votes are included in the proposed block.
2) The service consumers have to escrow deposit when they post a task, and the service providers get their share from the escrowed deposit after the successful completion of a task. When the service providers and consumers perform activities, their activity rate is incremented.

3) It selects a trusted leader from blockchain management nodes to propose a block and trusted validators from blockchain management nodes, crowdsourcing service providers, and consumers to validate and vote for the transactions and the block. The trusted leader and validators have to escrow deposit to prevent malicious behaviors.
4) Unlike the existing consensus protocols of blockchain-based crowdsourcing [18]–[21], the selection criteria of a leader and the validators are different. While the leader is selected based on three trust factors (i.e., deposit ratio, activity rate, missed rate), the validators are chosen based on deposit ratio and activity rate.

The STBC consensus protocol prevents blockchain fork, selfish mining, and invalid block insertion attacks. In STBC consensus protocol, at most one block is proposed in every round, which avoids the possibility of a blockchain fork. If multiple nodes propose a block simultaneously, then blockchain can fork. A single leader is selected to propose a block in each round of consensus to prevent this situation. A successfully generated block that is accepted by a majority of the validators is only added in the blockchain. In this way, all the nodes record the same block, which ensures the absence of a fork. Moreover, only the block of a trusted leader is added to the blockchain which prevents selfish mining attack. If a leader acts maliciously to add an invalid block, the consensus protocol resists it by adding no block.

The guarantee of correctness of any system is essential, especially critical and real-time systems. As a crowdsourcing system involves several critical tasks, therefore its correctness has vital importance. Formal methods are rigorous mathematical techniques to model computer systems that help designers verify the system's properties and assist in system testing to increase the confidence of correctness of a system. Model checking is an important formal methods-based technique used for the formal verification of critical systems. It is automatic, effective, and requires less human intervention. Therefore, it is utilized in this work.

The main contributions of this work are summarized below:

1) Security: The proposed consensus protocol provides security by defining security properties, i.e., safety and fault tolerance to prevent blockchain fork and selfish mining attacks. It is secure when at most $\left\lfloor \frac{n-1}{3} \right\rfloor$ nodes are faulty. Here n denotes the total number of nodes. This ratio is assumed similar to most of the byzantine fault tolerance (BFT) consensus protocols.
2) Trustworthy: We presented an improved trust model based on several new factors such as deposit ratio, activity rate, and missed rate. The leader and validators' trust properties are defined to prevent blockchain fork, selfish mining, and invalid block insertion attacks.
3) Fairness: The proposed consensus protocol provides a fair environment for the selection of leaders and validators. The *activity rate* is included to prevent rich nodes dominating the network. The introduction of *missed rate* of a node will increase its chances to be selected as a leader. The trust values update mechanism enforces nodes to obey protocol rules; because in the case of malicious activity, in addition to trust values, they also lose their deposit as a punishment.
4) Energy-saving: The protocol saves energy as a single leader is supposed to propose a block in every round.
5) Correctness: The correctness of the proposed consensus protocol is ensured utilizing a formal methods-based technique, i.e., model checking. The formal specification of the proposed STBC protocol is performed using CSP#. Furthermore, the PAT model checker is applied for the formal verification of safety, fault tolerance, leader trust, and validators' trust properties. Linear Temporal Logic (LTL) is utilized to specify the security and trust properties.

This work is novel in formal modeling and verification of the consensus protocol for a blockchain-based crowdsourcing system ensuring safety, fault tolerance, leader trust, and validators' trust. Albeit there are some formally verified blockchain consensus protocols [25]–[28], some of the existing consensus protocols for a blockchain-based crowdsourcing system are partially verified [18], [21]. This work is different from [25]–[28] as the proposed protocol is designed for a crowdsourcing application and in comparison to [18], [21] the proposed consensus protocol is formally verified.

The rest of the paper is organized as follows: Section II establishes the background knowledge; Section III discusses the related work; Section IV presents the system model; Section V defines the formal model of the proposed protocol; Section VI describes the results; and finally Section VII presents the conclusion and future work.

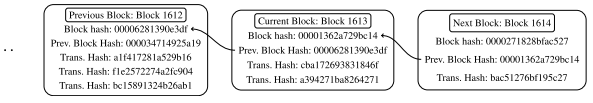

FIGURE 2. An example of blockchain.

II. BACKGROUND

This section provides background knowledge about blockchain and formal modeling and verification.

A. BLOCKCHAIN TECHNOLOGY

Blockchain technology was introduced by an unknown person *Satoshi Nakamoto*, as a distributed, peer-to-peer, decentralized, and a linked structure to address the issue of double-spending [13]. Figure 2 represents an example of blockchain. The transactions are grouped and ordered in a structure called a *block*. In addition to transactions, a block contains its hash and the previous block hash. The variable *Prev. Block Hash* in the current block in Figure 2 is used to link the block to its previous block which prevents alteration of blocks and insertion of a block between existing blocks. The miners are responsible for adding the blocks in

chronological order. In the blockchain, any node can initiate a transaction and broadcast it to the nodes of the network. The network nodes validate transactions and the miner of the block adds the validated transactions to the blockchain. The number of transactions in a block depends upon the size of a block.

Blockchain have applications in various fields, for example, healthcare [29], banking sector [30], insurance companies [31], and crowdsourcing [32]. In this work blockchain is exploited for crowdsourcing systems, as it helps to provide accountability, transparency, and immutability to avoid unfaithful and dishonest behaviors.

B. FORMAL MODELING AND VERIFICATION

Formal modeling and verification of consensus protocols is important to ensure their correctness. Model checking is the formal methods-based technique that is effectively used for formal verification. Model checking involves verifying whether a formal model of the system satisfies the desired properties. Model checking technique is useful to reveal the errors that cannot be identified using testing and simulation techniques. In this work, we use the model checking technique because in comparison to other techniques, e.g., theorem proving, it is more effective, automatic, and needs less human involvement. Model checking technique is useful to check partial specifications. However, the state explosion problem is a big limitation of the model checking approach [33], as the state of the model grows infinitely large with the increase of variables, distinct values, and components.

CSP# is a formal specification language used for the formal modeling of concurrent aspects of critical systems. The high-level modeling constructs such as sequential/parallel composition, channels, and interleaving are combined with low-level C# constructs, e.g., variables, if-then-else, and arrays. The syntax of CSP# can be described as:

$Proc ::= \text{SKIP}$	(termination)
$\mid \text{STOP}$	(deadlock)
$\mid ev\{program\} \rightarrow Proc$	(operation prefixing)
$\mid ev \rightarrow Proc$	(event prefixing)
$\mid [guard]\, Proc$	(state guard)
$\mid Proc_1 \mid\mid Proc_2$	(parallel composition)
$\mid Proc_1 \mid\mid\mid Proc_2$	(interleaving)
$\mid Proc_1 ; Proc_2$	(sequential composition)
$\mid ch?a \rightarrow Proc(a)$	(channel input)
$\mid ch!a \rightarrow Proc$	(channel output)
$\mid Proc_1 \,\Box\, Proc_2$	(external choice)
$\mid Proc_1 \sqcap Proc_2$	(internal choice)
$\mid \text{if } guard \text{ then } Proc_1 \text{ else } Proc_2$	(boolean statement)
$\mid Proc_1 \triangleright Proc_2$	(timeout)
$\mid Proc_1 \triangle Proc_2$	(interrupt)

where $Proc$, $Proc_1$ and $Proc_2$ are processes, *guard* represents a condition, *ev* is an event, *program* describes a piece of code that executes atomically and *ch* denotes a synchronized communication channel.

Assertions are used to query properties about the behavior of a system. Various types of assertions are supported in PAT, such as, deadlock-freeness, divergence-free, nonterminating, reachability and LTL. Deadlock-freeness and LTL assertions are used in this work.

C. DEADLOCK-FREENESS

This assertion checks whether a process *Proc* is deadlock-free or not.

#assert *Proc*() deadlockfree;

D. LINEAR TEMPORAL LOGIC (LTL)

LTL is used to specify properties to be checked on a formal model. LTL extends predicate or propositional logic by modalities and provide a way to mathematically represent linear time properties. PAT supports the full syntax of LTL. For a process *Proc*(), the below assertion checks whether *Proc*() satisfies the LTL formula ϕ.

#assert *Proc*() $\models \phi$;

The syntax of LTL is defined according to following rules:

$$\phi = ev \mid prop \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \Box\phi \mid \Diamond\phi \mid \bigcirc\phi \mid \phi_1 \cup \phi_2$$

where *ev* is an event, *prop* is an atomic proposition, $\Box$ is read as always, $\Diamond$ is read as eventually, $\bigcirc$ is read as next, and $\cup$ is read as until. An LTL formula $\Box\phi$ defines that ϕ must hold on the entire respective path while $\Diamond\phi$ represents that ϕ eventually holds on the respective path. An LTL formula $\bigcirc\phi$ describes that ϕ must hold at next state whereas $\phi_1 \cup \phi_2$ shows that ϕ_2 holds at the current position or at some future position and ϕ_1 has to hold before that state. From that position ϕ_2 does not necessarily has to hold any more.

III. RELATED WORK

In this section, firstly blockchain consensus protocols are described, then blockchain-based crowdsourcing consensus protocols are discussed and crowdsourcing systems based on blockchain are described in the end.

A. BLOCKCHAIN CONSENSUS PROTOCOLS

Some important blockchain consensus protocols are briefly described here. PoW is the first consensus algorithm that is used in Bitcoin [13]. It is based on complicated computation where every node of the network continuously computes a hash value of every block header using different nonce values until the calculated hash value becomes less than or equal to a certain target value. When a node gets the desired value, other nodes of the network verifies its correctness, and the block is added to the blockchain. However, it is highly resource-intensive and has low throughput.

PoS is an alternative to PoW that requires much less energy to be consumed [14]. In this mechanism, a node is selected based on its stake to propose a block. However, it favors rich nodes in the network. In this way, centralization can occur in the network. Several solutions are proposed to address this issue. For example, Peercoin [34] uses coin age selection procedure, Blackcoin [35] is based on randomization and Bitshares [36] uses Delegated Proof of Stake (DPoS) [15] in

which stakeholders select delegates based on voting for the generation and validation of a block, however, it has some degree of centralization.

PBFT is proposed for tolerating byzantine faults [37]. It can tolerate 1/3 byzantine faults. The consensus is divided into three phases, i.e., pre-prepare, prepared, and commit. A node can transit from one phase to another phase after getting 2/3 votes from other nodes. Tendermint is another BFT consensus protocol [38]. However, the BFT protocols are suitable for small networks and have poor scalability. The survey of several other blockchain consensus protocols is provided in [39].

B. BLOCKCHAIN-BASED CROWDSOURCING CONSENSUS PROTOCOLS

Recently, the integration of blockchain technology and crowdsourcing has gained the attention of the research community. PoT is a consensus protocol built upon the idea of blockchain technology to handle the accountability issue in crowdsourcing services [18]. The protocol utilizes RAFT [22] to elect a leader from ledger nodes and selects service participants as validators of transactions based on trust values. The PoT protocol addresses the issue of scalability associated with BFT and Paxos-based algorithms. It considers the unfaithful behavior of nodes that is not addressed in BFT algorithms. The security attacks, including, sybil, collusion, and DDoS are analyzed in the work. As RAFT is formally verified through TLA proof system and Coq proof assistant [22], [23], therefore, PoT is defined as partially verified in Table 1. However, PoT cannot preserve privacy of users. An improved PoT consensus algorithm is presented for crowdsourcing using blockchain as an underlying technology [19]. In this protocol, a high reputation worker is selected as a miner of the block. The calculation of reputation is based on the historical interaction of a miner and feedback of other nodes which may not be reliable. It analyzes the collusion attacks and does not provide a mechanism to evaluate task solutions. Further, privacy and reliability are also not addressed in this work.

MCS-Chain is presented as a blockchain-based mobile crowdsourcing platform for decentralized and distributed tr-ust management [20]. The trust evaluation mechanism is defined which allows users to select reliable workers. A consensus protocol is presented in MCS-Chain for block generation to improve the efficiency of the blockchain. In the protocol, a new block is created when the total payment which needs to be recorded into the next block exceeds the defined threshold. The centralization and fork issues are addressed in the work which guarantee that even if several blocks are generated simultaneously, a unique block is added to the blockchain. The liveness, safety, fault tolerance, and decentralization are demonstrated theoretically but the system lacks in privacy and anonymity of users and workers.

The zkCrowd, a hybrid blockchain-based crowdsourcing platform is proposed to balance transaction transparency and privacy [21]. The zkCrowd consists of a public chain to manage public tasks and multiple subchains to handle private tasks. DPoS is implemented on the public blockchain to elect validators. The alternative validators which are not elected in the public chain are elected as subchain validators in a round-robin manner using PBFT on private subchains. The zkCrowd is resilient against a 51% attack, sybil attack, privacy leakage, free riding and false reporting, and byzantine failures. As continuous-time Markov chain models are defined for PBFT and are formally verified through PRISM model checker [24], therefore, zkCrowd is termed as partially verified in Table 1. However, zkCrowd lacks in defining cross-communication between public and private chains, privacy protection of answers, and a proper reward distribution mechanism. On the public chain, voters having more tokens have more influence on the network. Further, if there are a large number of private tasks then an issue of scalability can occur. The comparison of the above-discussed consensus protocols is provided in Table 1. Unlike [18]–[21], the proposed protocol is formally verified.

TABLE 1. Summary of blockchain-based crowdsourcing consensus protocols.

Protocol / Platform	Limitations	Resilient attacks	Security properties	Trust properties	Formally verified
PoT [18]	Cannot preserve privacy of users	Sybil, Collusion and DDoS	Validity, Agreement, Liveness, Fault tolerance, Accountability	Validators trust	Partially
Improved PoT [19]	It is reputation based which depends on opinions and sometimes cannot be reliable	Collusion attacks	Authentication, Safety, Accountability	Leader and Validators trust	No
MCS-Chain [20]	Lacks in preserving privacy. Security properties are described theoretically	Centralization and fork issues	Integrity, Liveness, Safety, Fault tolerance, Decentralization	Leader and Validators trust	No
zkCrowd [21]	Lacks in defining cross-communication between public and private chains. Voters having more tokens have more influence on the network.	51%, Sybil, Privacy leakage, Free riding, false reporting and Byzantine failure	Authentication, Non-repudiation, Fault tolerance, Fraud resilience, Accountability	-	Partially

C. BLOCKCHAIN-BASED CROWDSOURCING SYSTEMS

1) TRUST MANAGEMENT

MCS-Chain+ is presented to address the issue of privacy in MCS-Chain [40]. The Intel Software Guard Extension (SGX) is utilized to anonymously authenticate trust through a trustworthy evaluation of trust. RC-chain is a reputation-based framework that is proposed as a crowdsourcing blockchain to support crowdsourcing trading and evaluation of user-reputation [41]. CrowdR-FBC is presented as a distributed fog-blockchain mechanism for reputation management in crowdsourcing which prevents leakage of users' privacy, involvement of malicious users, and tampering of users' reputation [42]. It ensures the selection of trusted workers and the reliability of data. Blockchain technology is utilized to present a trusted and decentralized approach for a crowdsourcing system [43]. The proposed mechanism enables information of users and several behaviors to be transparent and cannot maintain the privacy of user's information. A blockchain-based trust model using weighted consensus technique to share information in crowd environments is presented to achieve higher accuracy [44].

TABLE 2. Comparison of blockchain-based crowdsourcing systems.

		Security													Trust				
References	Category	CP	C	I	AT	AZ	AV	NR	P	LN	S	FT	FR	AC	LT/MT	VrT/VlT	WT	DT	PDT
[18]	Trust management	✓	×	×	×	×	×	×	×	✓	×	✓	×	✓	×	✓	×	×	×
[19]		✓	×	×	✓	×	×	×	×	×	✓	×	×	✓	✓	✓	✓	×	✓
[20]		✓	×	✓	×	×	×	×	×	✓	✓	✓	×	×	✓	✓	✓	✓	✓
[21]		✓	×	×	✓	×	×	✓	×	×	×	✓	✓	✓	×	×	×	×	×
[40]		✓	×	✓	✓	×	×	×	×	×	×	×	×	×	✓	✓	✓	×	×
[41]		✓	×	×	✓	✓	×	×	×	×	×	×	×	×	×	×	×	✓	×
[42]		×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	✓	✓	×
[43]		×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	✓	×	×
[44]		×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	✓	×	×
[45]		✓	×	×	×	×	×	×	×	×	×	×	✓	✓	✓	✓	✓	✓	✓
[46]		✓	✓	×	×	×	×	×	×	×	×	×	×	✓	✓	✓	×	×	×
[32]	Task management	×	×	✓	✓	×	×	×	×	×	×	×	✓	×	×	×	✓	✓	✓
[47]		×	×	×	✓	×	×	×	×	×	×	×	×	✓	×	×	×	×	×
[48]		×	✓	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
[49]		×	×	×	×	✓	×	×	×	×	×	×	×	×	×	×	×	×	×
[50]		×	×	✓	×	×	×	×	×	×	×	×	✓	×	×	×	×	×	×
[51]		×	×	✓	×	×	×	×	×	×	×	×	✓	×	×	×	×	×	×
[52]		×	×	✓	×	×	×	×	×	×	×	×	✓	×	×	×	×	×	×
[53]		×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
[54]		×	×	×	✓	×	×	×	×	×	×	×	×	×	×	×	×	×	×
[55]		×	×	×	✓	×	×	×	×	×	×	×	×	×	×	×	✓	×	×
[56]		×	✓	✓	✓	×	×	×	×	×	×	×	✓	×	×	×	×	×	×
[57], [58]		×	×	✓	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
[59]	Dispute arbitration	×	✓	✓	✓	✓	×	×	×	×	×	×	✓	×	×	×	×	✓	✓
[60]		×	×	×	×	×	×	×	×	×	×	×	✓	×	×	×	✓	✓	×
[61]		×	×	×	×	×	×	✓	×	×	×	×	✓	×	×	×	✓	✓	×
STBC		✓	×	×	×	×	×	×	×	×	✓	✓	×	×	✓	✓	×	×	×

CP : Consensus Protocol
C : Confiendtiality
I : Integrity
AT : Authentication
AZ : Authorization
AV : Availability
NR : Non-Repudiation
P : Persistence
LN : Liveness
S : Safety
FT : Fault Tolerance
FR : Fraud Resilience
AC : Accountability
LT : Leader Trust
MT : Miner Trust
VrT : Verifier Trust
VlT : Validator Trust
WT : Worker Trust
DT : Data Trust
PDT : Personal Data Trust
✓ : satisfies a property
× : does not satisfy a property

2) TASK MANAGEMENT

CrowdBC is a blockchain-based decentralized framework for solving tasks of requesters by a crowd of workers without any trusted third party [32]. It protects the privacy of users as it utilizes pseudonymous addresses for the identities of requesters and workers, and stores encrypted solutions in distributed storage. However, defining an efficient evaluation function for tasks solution is crucial in CrowdBC. ZebraLancer is another platform that protects user privacy, but it depends on a trusted third party for registration of identities [47]. A novel hybrid blockchain-based crowdsourcing platform is proposed to ensure decentralization and privacy [45]. The platform consists of a hybrid structure of blockchain and dual consensus protocols to ensure secure communication among requesters and workers. Zero-knowl-edge proofs are used to protect the privacy of users whereas the access control model is not defined in detail. FedCrowd platform is proposed as a privacy-preserving and federated platform for blockchain-based crowdsourcing [48]. The sma-rt contracts are employed as a trusted platform to publish encrypted tasks and craft matching protocols are designed for task recommendation in an efficient way. But it depends on a trusted third party for the management of public and private keys. PFcrowd platform is presented to allow crowdsourcing systems to perform encrypted task-worker matching on the blockchain without any third party [49].

Blockchain-powered crowdsourcing model for the mobile environment is proposed to address several challenges such as, participants privacy, the integrity of services, and improving quality of experience [50]. A blockchain-based task matching (BPTM) model for crowdsourcing for reliable and secure matching is proposed using smart contracts [51]. Confidentiality and identity anonymity are achieved using searchable encryption. An auction-blockchain-based crowdsourcing technique, ABCrowd is presented to execute crowdsourcing on Ethereum blockchain including auctions [52]. It utilizes repeated bidders auction technique which allows truthful bidding. However, it lacks to protect user privacy. Task Select Worker Crowd (TSWCrowd) is presented to address the issues of reliable tasks allocation, ensuring workers' payments and reliability of the platform [53]. The task-select-worker mechanism is defined to sort the tasks and the tasks having higher priority are assigned first. In contrast to ABCrowd, the average workers' payment in TSW-Crowd

is higher. ConGradetect is presented as a blockchain-based crowdsourcing detection system that addresses code and identity privacy but it analyzes simple codes [54].

Blockchain-based crowdsourcing is used for reliable data analysis in Mobile Ad-hoc Cloud (MAC) [55]. It addresses the challenges of attracting mobile devices to join the MAC network and able to collect reliable computation from mobile devices. A resilient-improved two-stage auction (ITA) is presented for fault tolerance in mobile crowdsourcing [56]. The proposed technique is beneficial for organizations as the workers are selected according to the time and budget of organizations. The designs for crowdsourced energy systems using blockchain technology are presented in [57], [58] to crowdsource small-scale production or trading of energy from distributed resources of energy. To handle the issues of crowdsourcing data access, analysis, and management, a framework based on blockchain technology is proposed [46]. The framework helps researchers in accessing operational data ensuring confidentiality, traceability, and accountability of data, and disseminating data in a controlled way to the public, but it lacks scalability.

3) DISPUTE ARBITRATION

A blockchain-based scheme for fine-grained authorization in data crowdsourcing (BC-FGA-DCrowd) is proposed in [59]. The scheme resists internal malicious actions and external sybil and DDoS attacks. The feasibility of the scheme is tested on the Ethereum network, however, it lacks in defining a proper function for evaluating the quality of data. A decentralized oracle-based game is proposed to decide the truth of queries and to resolve disputes [60]. In crowdsourcing, the quality of completed tasks and fairness in the evaluation of tasks are addressed using blockchain technology [61]. An arbitration mechanism is described for the realization of business services and to avoid security attacks, e.g., false reporting and free-riding but it lacks privacy analysis.

The comparison of the discussed research work in terms of security and trust is provided in Table 2. It is observed in the literature that there are very few blockchain-based crowdsourcing consensus protocols and some of these are partially verified [18], [21]. To the best of our knowledge, this is the first work which utilizes CSP# for the formal specification of a blockchain-based crowdsourcing consensus protocol and the PAT model checker for the verification of security and trust properties.

IV. SYSTEM MODEL AND CONSENSUS PROTOCOL

This section describes the architecture of the system, design of the consensus protocol, security properties, and associated attacks. In the end, a threat model with attacks and defenses is presented.

A. SYSTEM ARCHITECTURE

The architecture of the blockchain-based crowdsourcing system (BBCS) is represented in Figure 3. It consists of four

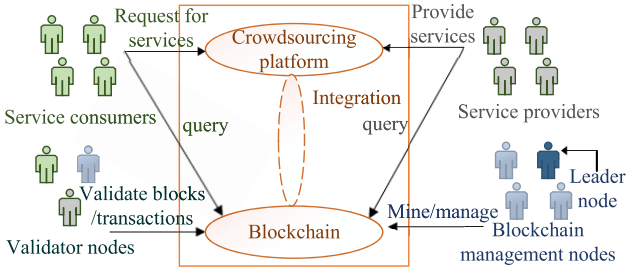

FIGURE 3. Architecture of the system.

actors: service consumer, service provider, blockchain management nodes, and validator nodes. The service consumers request for services by posting a task to the crowdsourcing platform. A service consumer can check the status of a task by sending a query to the blockchain. Service providers provide services by receiving a task through a crowdsourcing platform. A service provider can send a query to the blockchain to get information about the task payment. The blockchain management nodes are responsible for managing blockchain, e.g., proposing and validating blocks. In every round of consensus, a trusted leader is elected from the blockchain management nodes to propose and add a block to the blockchain. The validators validate and vote for transactions and blocks. The blockchain management nodes, service consumers, and providers can serve as validators.

B. DESIGN OF CONSENSUS PROTOCOL

1) CROWDSOURCING CONSENSUS PROTOCOL

The overview of the proposed STBC consensus protocol is presented in Figure 4. The consensus process starts with the initialization of nodes that can perform the transactions. The blockchain management node having the highest trust value is selected as a leader for the current round. The trust is calculated based on deposit ratio, activity rate, and missed rate.The trusted validators are selected from blockchain management nodes, crowdsourcing service providers, and consumers to validate and vote for the transactions and the block to avoid malicious behaviors. The validator's trust is calculated based on deposit ratio and activity rate. The process of selection of leader and validators runs in parallel to save time. After the process of selection, the trusted validators vote for transactions and broadcast votes. The leader prepares the block by including the transactions that gain the majority of the votes from validators. The leader then broadcasts the block to the validators and they vote for the block. If the majority of the validators accept the block then the leader adds the block in the blockchain and all the nodes update their ledger. The trust values and deposit of leader and validators are updated and the consensus process enters into the next round. If the block is rejected, then the leader and validators lose the trust and deposit, and the consensus protocol moves to the next round.

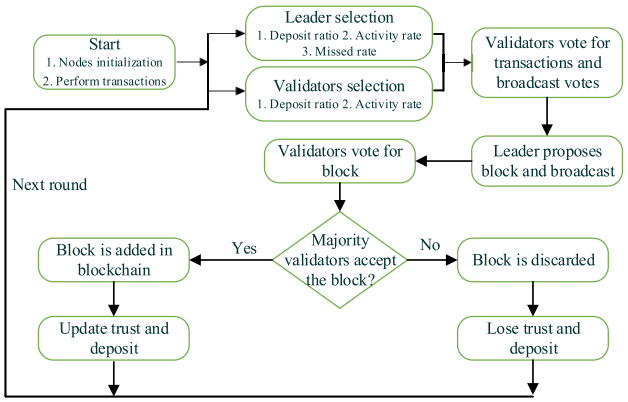

FIGURE 4. Overview of STBC consensus protocol.

2) CALCULATION OF TRUST OF A LEADER

The trust factors of management nodes for the selection of a leader are defined as:

Deposit ratio of a management node: The deposit that a management node i made is denoted as md_i. The sum of deposits of all management nodes is defined as $MD = \sum md_i$. The deposit ratio of a management node i is calculated as $\alpha_i = \frac{md_i}{MD} \in [0, 1]$. All the nodes should have the deposit greater than some specific amount for the block generation process.

Activity rate of a management node: Let MB_i be the total number of times that a management node i has been selected for block generation and mb_i be the number of times that a management node i has successfully generated the block. The activity rate of a management node is denoted as $\beta_i = \frac{mb_i}{MB_i} \in [0, 1]$. Initially, it is assumed that $\beta_i = 0$.

Missed rate of a management node: Let MS_i be the total number of times that a management node i participates in leader selection and ms_i be the number of times that a management node i has not been selected as a leader. The missed rate of a management node is calculated as $\gamma_i = \frac{ms_i}{MS_i} \in [0, 1]$. It is assumed that $\gamma_i = 0$ initially.

A leader for block generation is selected from management nodes based on the above trust factors.

Trust of a leader: The trust of a leader i can be calculated as: $\eta_i = \frac{\alpha_i+\beta_i+\gamma_i}{3} \in [0, 1]$. The deposit ratio can affect the leader selection process. Therefore, to avoid the situation where a malicious attacker can dominate the network by submitting a tremendous amount of money, a maximum deposit that a node can submit is defined, which is agreed by all participants. In the first few rounds, the nodes having higher deposit and activity rate have a greater chance to be selected as a leader but in the long-term the nodes having a deposit and higher missed rate have an equal chance to be selected as the nodes having higher deposit and activity rate. Missed rate is introduced to provide fairness among nodes.

3) CALCULATION OF TRUST OF VALIDATORS

The trust factors to define a criteria for the selection of validators are described below.

Deposit ratio of a validator node: The deposit that a validator node j made is denoted as vd_j. The sum of deposits of all validator nodes is defined as $VD = \sum vd_j$. The deposit ratio of a validator node j is calculated as $\Gamma_j = \frac{vd_j}{VD} \in [0, 1]$. All the nodes should have a deposit greater than some specific amount for the block validation process.

Activity rate of a validator node: Let VB_j be the total number of times that a validator node j has been selected for block validation and vb_j be the number of times that a validator node j has successfully validated the block. The activity rate of a validator node is denoted as $\Lambda_j = \frac{vb_j}{VB_j} \in [0, 1]$. Initially, it is assumed that $\Lambda_j = 0$.

Trust of a validator: The trust of a validator j can be calculated as: $\sigma_j = \frac{\Gamma_j+\Lambda_j}{2} \in [0, 1]$. The activity rate is introduced here to avoid rich nodes dominating the network.

4) ALGORITHM

The high-level pseudocode of STBC consensus protocol is described in Algorithm 1. It takes management M, consumer C, provider P and validator V nodes as input (line 1). Firstly, all these nodes are initialized (line 2). The consumer and provider nodes can perform transactions (line 3). These two steps are required for running the consensus protocol. Then the selection of a trusted leader *TLeader* and K trusted validators *TValidators* runs in parallel (lines 4-5). After the selection process, the trusted validators vote for T transactions for a block (lines 6-7). The leader proposes the block by including the leader information, previous block hash, and T transactions that receive maximum votes (lines 8-9). The proposed block is then broadcast for a vote to validators (lines 10-11). If it is accepted by the majority of the validators then it is added in the blockchain and the leader and validators' trust values and deposit are updated (lines 12-14). Otherwise, the proposed block is discarded and the selected leader and validators lose the trust and escrow deposit (lines 15-17). The number of activities $mb_{TLeader}$, number of misses $ms_{TLeader}$ and deposit $md_{TLeader}$ of the leader are set to zero. The number of activities $vb_{TValidators}$ and deposit $vd_{TValidators}$ of trusted validators become zero. The whole process is repeated in the next round (line 18). In Algorithm 1, *-Blockchain* represents the old state of blockchain and ^ is the concatenation operator.

The high level pseudocode of *Nodes_Initialization* is presented in Algorithm 2. A management node i is defined by its deposit md_i, number of activities mb_i, and number of misses ms_i. Suppose a management node i has a deposit that lies between the minimum and maximum deposit range. If the number of activities and misses are initially zero, then it can become part of the network (lines 2-5). It can serve as a validator, therefore, it is added in the validator nodes (line 6). Similarly, suppose a consumer node c and a provider node p have sufficient deposit and zero number of activities. In that case, they can be added to the network (lines 7-10, 12-15). A consumer and a provider can also serve as a validator, therefore, included in the validator nodes

Algorithm 1: High level pseudocode of STBC consensus protocol.

```
1  STBC(M, C, P, V)
2    Nodes_Initialization(M, C, P, V)
3    Perform_Transactions(C, P)
4    Selection() = Leader_Selection(M) ||
5                  Validators_Selection(V, K)
6    VTransactions ← VoteBroadcast(Transactions, T,
7                    TValidators)
8    Propose(Block(index, TLeader, prevHash, hash,
9            Tmax(VTransactions)))
10   Broadcast(Block, TValidators)
11   Vote(Block, TValidators)
12   if (Accept(Block, Majority) == true)
13     Blockchain ← -Blockchain ^ Block
14     UpdateTrustAndDeposit()
15   else
16     Discard(Block)
17     LoseTrustAndDeposit()
18   Next_Round()
```

(lines 11, 16). Here, s, t, and u represent a number of management, consumer, and provider nodes respectively that can be added to the network.

Algorithm 2: Pseudocode of nodes intialization.

```
1  Nodes_Initialization(M, C, P, V)
2    for i = 1, 2, . . . , s
3      if(md_i ≥ minDeposit && md_i ≤ maxDeposit &&
4         mb_i == 0 && ms_i == 0)
5         M = M ∪ {i}
6         V = V ∪ {i}
7    for c = 1, 2, . . . , t
8      if(vd_c ≥ minDeposit && vd_c ≤ maxDeposit
9         && vb_c == 0)
10        C = C ∪ {c}
11        V = V ∪ {c}
12   for p = 1, 2, . . . , u
13     if(vd_p ≥ minDeposit && vd_p ≤ maxDeposit
14        && vb_p == 0)
15        P = P ∪ {p}
16        V = V ∪ {p}
```

Algorithm 3 defines the types of transactions that can occur in a crowdsourcing system. A consumer c can post a task and the deposit is stored in blockchain as an escrow amount (lines 2-4). A consumer deposit includes three shares, the first share is for task processing, the second share is the block reward and the third share is the verification fee. The number of activities of a consumer is updated (line 5). A provider p can receive and perform a task (lines 6-8). The consumer then evaluates the task and if it is accepted then the provider gets the share of payment and the number of activities is incremented (lines 9-13). Otherwise, the consumer gets some proportion of the deposit (lines 14-17).

Algorithm 3: Pseudocode of performing transactions.

```
1  Perform_Transactions(C, P)
2    ∃ c ∈ C
3      PostTask(c)
4      Escrow(d_c)
5      c.UpdateNumActivities(1)
6    ∃ p ∈ P
7      ReceiveTask(p)
8      PerformTask(p)
9      EvaluateTask(c)
10     if (AcceptTask(c) == true)
11       treward ← RemoveEscrow(d_c/total_shares)
12       TaskReward(p, treward)
13       p.UpdateNumActivities(1)
14     else
15       RejectTask(c)
16       pdeposit ← RemoveEscrow(d_c/total_shares)
17       DepositReturned(pdeposit)
```

Algorithm 4 describes the procedure of leader selection (line 1). Initially, the first management node M_0 is selected as a temporary trusted leader (line 2). The participation values $MS_{TLeader}$ and MS_x of all the management nodes that take part in leader selection are incremented (lines 3, 6). From all the management nodes, a management node having the highest trust value η is selected as a trusted leader (lines 4-5, 7-11). In case of a tie, an approach presented in [62] is utilized to avoid it. According to this, a management node having the smallest hash value of its identifier (or public key) and trust is selected (lines 12-18). The selection value of the trusted leader $MB_{TLeader}$ is incremented (line 19). The deposit of the selected leader is stored as an escrow amount to prevent malicious behavior (line 20).

Algorithm 5 presents the process of selection of K validators. The value for K can be selected at the time of implementation. Initially, no trusted validators are selected (line 2). The validators are sorted in descending order according to the trust value σ, and the first K validators of highest trust values are selected (lines 3-12). It is ensured that a leader cannot serve as a trusted validator in the same round. The selection value VB_{V_a} of every trusted validator is updated (line 13). The deposit vd_{V_a} of each selected validator is stored in block-chain to prevent malicious behavior (line 14).

The update trust process is described in Algorithm 6. When the leader's block is accepted by the network, then its number of activities $mb_{TLeader}$ is incremented (line 2), it gets the block generation reward (lines 3-4) and gains its escrowed deposit (line 5). The number of misses $ms_{M\setminus TLeader}$ of management nodes other than the leader node is updated (line 6). The number of activities $vb_{TValidators}$ of trusted validators is incremented (line 7). The validators get the

Algorithm 4: Pseudocode of leader selection.

```
1  Leader_Selection(M)
2    let TLeader = M_0
3    MS_TLeader = -MS_TLeader + 1
4    x = 1
5    while (x < len M)
6      MS_x = -MS_x + 1
7      if (η_TLeader > η_Mx)
8        x++
9      else if (η_TLeader < η_Mx)
10       TLeader = M_x
11       x++
12     else if (η_TLeader == η_Mx)
13       if (Hash(TLeader.GetMId(), η_TLeader) >
14         Hash(M_x.GetMId(), η_Mx))
15         TLeader = M_x
16         x++
17       else
18         x++
19   MB_TLeader = -MB_TLeader + 1
20   Escrow(md_TLeader)
```

Algorithm 5: Pseudocode of validators selection.

```
1  Validators_Selection(V, K)
2    len(TValidators) = 0
3    a = 0, b = 1
4    while (a < len V − 1 && V_a != TLeader &&
5     len(TValidators) ≤ K)
6      while (b < len V && V_b != TLeader)
7        if (σ_Va ≥ σ_Vb)
8          b++
9        else
10         Swap(V_a, V_b)
11         b++
12     TValidators = TValidators ∪ V_a
13     VB_Va = -VB_Va + 1
14     Escrow(vd_Va)
15     a++
```

block validation reward (lines 8-9). The escrowed deposit of validators is returned (line 10).

C. PROPERTIES AND ATTACKS

In this work, we have defined security and trust properties, i.e., safety, fault tolerance, trusted leader, and trusted validators to prevent security attacks, such as blockchain fork, selfish mining, and invalid block insertion. The properties and attacks are defined below.

Definition 1 (Safety): The consensus protocol is safe if all the nodes of network eventually reach the same decision.

Algorithm 6: Pseudocode of updating trust and deposit.

```
1  UpdateTrustAndDeposit()
2    mb_TLeader = -mb_TLeader + 1
3    greward ← RemoveEscrow(d_c/total_shares)
4    BlockGReward(greward)
5    DepositReturned(RemoveEscrow(md_TLeader))
6    ms_M\TLeader = -ms_M\TLeader + 1
7    vb_TValidators = vb_TValidators + 1
8    vreward ← RemoveEscrow(d_c/total_shares)
9    BlockVReward(TValidators, vreward)
10   DepositReturned(RemoveEscrow(vd_TValidators))
```

Definition 2 (Fault Tolerance): The consensus protocol is fault tolerant if all the honest nodes eventually accept the same block even in the presence of malicious nodes.

Definition 3 (Trusted Leader): In every round of consensus protocol, a management node having the highest trust value is selected eventually.

Definition 4 (Trusted Validators): In every round of consensus protocol, validator nodes that have the highest trust values are eventually selected.

Definition 5 (Blockchain Fork): It represents a state of the blockchain in which nodes have different views about the blockchain state. The network nodes may receive more than one block at the same time.

Definition 6 (Selfish Mining): Miners secretly mine blocks to earn rewards. The selfish miners release the blocks to the public when they get a longer chain.

Definition 7 (Invalid Block Insertion): The blocks that disobey network rules are known as invalid blocks. A malicious leader tries to add invalid blocks in the blockchain.

D. THREAT MODEL

To introduce an effective attack against the STBC consensus protocol, an attacker needs to compromise more than 1/3 of network nodes. A malicious leader and a validator can pose several threats which can affect the working of the consensus process. If there are multiple nodes mining at the same time then blockchain can fork, therefore, in the proposed consensus protocol, a unique leader is selected in every round to propose a block to prevent blockchain fork. Malicious nodes may try to mine blocks secretly and broadcast to the network nodes. To avoid this situation, it is ensured that only the trusted leader can execute the block propose and broadcast process and all the nodes add the block of a trusted leader in their ledger in each round. A malicious leader may try to add an invalid block in the blockchain, but the consensus protocol prevents its addition by adding no block. The deposit of a leader is stored in the blockchain to avoid malicious behavior. If a leader remains honest then the escrowed deposit is returned, otherwise, the deposit is slashed away. Moreover, it also loses its trust and it becomes very difficult for the

malicious leader to become the leader again. This incentivises the leader to behave honestly.

Malicious validators may refuse to vote for transactions or blocks to prevent the addition of the correct blocks. This situation can occur only if more than 1/3 of the validators are malicious. To avoid this situation, the selected validators must also submit a deposit to the blockchain. If the correct block cannot be added to blockchain due to malicious validators then the validators not only lose the trust but also the escrowed deposit which incentivises validators to behave honestly. Further, it becomes difficult for these nodes to become trusted validators again. Further, if the consensus process is localized among some fixed nodes then it can lead to centralization. Therefore, the trust model for the selection of a leader and validators is designed in such a way that it prevents centralization. For the selection of a trusted leader and validators, activity rate is utilized in addition to deposit ratio to restrict rich nodes from dominating the network. Missed rate is also introduced in the selection of a leader so that the nodes waiting to become leaders also get a chance. This ensures fairness and decentralization in the protocol.

V. FORMAL SPECIFICATION OF CONSENSUS PROTOCOL

This section presents the formal model of the STBC consensus protocol using CSP#. In the formal model, the structure of blocks and the blockchain is defined at an abstract level and our main focus is to model the communication among nodes to achieve the consensus.

A. CONSENSUS PROTOCOL

The consensus protocol is defined as a process (line 3) which involves four nodes represented as *PNodes* (line 1) where node $i \in PNodes$. The number of transactions to be included in a block is defined as T (line 2). All the steps of the consensus protocol are described in detail. In a distributed environment, all the nodes execute a sequence of processes of consensus protocol in parallel.

1. *#define PNodes* 4;
2. *#define T* 2;
3. *Consensus_ Protocol*() =
   (|| $i : \{0..PNodes-1\}$@
   (*NodeInitialization*(); *PerformTransactions*();
   Selection(); *VoteForTransactions*(T, i);
   ProposeAndBroadcast(i); *VoteToBlock*(i);
   isValidBlockchain(); *AddBlock*(i);
   UpdateTrustAndDeposit()));
   NextRound();

B. NODES INITIALIZATION

Firstly, the nodes are initialized having unique identifiers, with sufficient deposit, number of activities, and number of misses (line 4). There are three types of nodes, i.e., management nodes, provider nodes, and consumer nodes. All these nodes can serve as validator nodes.

4.*NodeInitialization*() = (*MngNode*1(0, 20, 0, 0); *MngNode*2(1, 30, 0, 0); *ProvNode*1(2, 15, 0); *ProvNode*2(3, 20, 0); *ConsNode*1(4, 10, 0); *ConsNode*2(5, 30, 0); *ConsNode*3(6, 20, 0));

*MngNode*1 is defined as a management node (line 5). The first parameter represents the identifier of the node. The deposit, number of activities, and number of misses are described in the second, third, and fourth parameters respectively. Firstly, a guard statement is specified in which it is checked that the node has a positive identifier, has sufficient deposit, and that the number of activities and misses lies between the minimum and maximum range (line 5.1). If the guard evaluates to true then the new management node is added (line 5.2). As a management node can serve as a validator, therefore, it is also included in the validator nodes. The behavior of *MngNode*2 is similar to *MngNode*1, therefore it is marked as "..." for simplicity (line 6).

5. *MngNode*1(*id*, *deposit*, *numActivities*, *numMisses*) =
5.1. [*id* >= 0 && *deposit* >= *minDeposit* &&
   deposit <= *maxDeposit* &&
   numActivities >= *minNumActivities* &&
   numActivities <= *maxNumActivities* &&
   numMisses >= *minNumMisses* &&
   numMisses <= *maxNumMisses*]
5.2. *add*1 {*var* m1 = *new ManagementNode*(*id*, *deposit*, *numActivities*, *numMisses*);
   managementNodes.Add(m1);
   validatorNodes.AddNode(*new ValidatorNode*(*id*, *deposit*, *numActivities*, *numMisses*, m1)); } → *Skip*;
6. *MngNode*2(*id*, *deposit*, *numActivities*, *numMisses*) = ...

*ProvNode*1 is described as a provider node (line 7). It is defined by an identifier, deposit, and number of activities. Provider nodes can also participate in validation. *ProvNode*2 (line 8) is similar to *ProvNode*1.

7. *ProvNode*1(*id*, *deposit*, *numActivities*) =
7.1. [*id* >= 0 && *deposit* >= *minDeposit* && *deposit* <= *maxDeposit* && *numActivities* >= *minNumActivities* && *numActivities* <= *maxNumActivities*]
7.2. *add*3 { *var* v1 = *new ProviderNode*(*id*, *deposit*, *numActivities*); *providerNodes.Add*(v1); *validatorNodes.AddNode*(*new ValidatorNode*(*id*, *deposit*, *numActivities*, v1)); } → *Skip*;
8. *ProvNode*2(*id*, *deposit*, *numActivities*) = ...

*ConsNode*1, *ConsNode*2 and *ConsNode*3 are defined as consumer nodes similar to provider nodes (lines 9-11).

9. *ConsNode*1(*id*, *deposit*, *numActivities*) = ...
10. *ConsNode*2(*id*, *deposit*, *numActivities*) = ...
11. *ConsNode*3(*id*, *deposit*, *numActivities*) = ...

C. PERFORM TRANSACTIONS

When the nodes become part of the system they perform transactions for the execution of tasks (line 12). The transactions of a post, receive, perform, evaluate, accept, reject, and finish a task are described.

12. *PerformTransactions*() =
   PostTask(*consumerNode*, 30); *ReceiveTask*(*providerNode*); *PerformTask*(*providerNode*); *EvaluateTask*(*consumerNode*); (*AcceptTask*(*consumerNode*) □ *RejectTask*(*consumerNode*));
   FinishTask(*providerNode*);

The post task transaction is specified as a process that takes a node *vl* and *deposit* as input (line 15). The input node should be a consumer node and initially, its state should be *not_post*. The deposit should exist between the minimum and maximum range (line 15.1). If the guard condition is satisfied then the state of the task changes to *post*. The consumer escrows the deposit in the blockchain. The number of consumer activities is updated (lines 15.2).

13. *var state*=[*not_post*, *post*, *receive*, *perform*, *evaluate*, *taccept*, *treject*, *finish*];
14. *var task_state* = *state*[0];
15. *PostTask*(*vl*, *deposit*) =
15.1. [*vl* == *consumerNode* && *task_state* == *state*[0] && *deposit* >= *minDeposit* && *deposit* <= *maxDeposit*]
15.2. *post_task* {*task_state* = *state*[1]; *escrowC* = *consumerNode*.*SubmitDeposit*(*deposit*); *blockchain*.*SetEscrowAmount*(*escrowC*); *consumerNode*.*UpdateNumActivities*(1); } → *Skip*;

When the task is posted it is received by the provider and the state changes from *post* to *receive*. This is described by the process *ReceiveTask* (line 16).

16. *ReceiveTask*(*vl*) = [*vl* == *providerNode* && *task_state* == *state*[1]]
16.1. *receive_task* {*task_state* = *state*[2]; } → *Skip*;

When the provider receives the task then he performs the task and the state of the task changes to *perform* (line 17).

17. *PerformTask*(*vl*) = [*vl* == *providerNode* && *task_state* == *state*[2]]
17.1. *perform_task* {*task_state* = *state*[3]; } → *Skip*;

After the task is performed by the provider, then the consumer evaluates it (line 18).

18. *EvaluateTask*(*vl*) = [*vl* == *consumerNode* && *task_state* == *state*[3]]
18.1. *evaluate_task* {*task_state* = *state*[4]; } → *Skip*;

The consumer can accept or reject the task (lines 19-20). If it is accepted then its state changes to *accept* (line 19.1).

19. *AcceptTask*(*vl*) = [*vl* == *consumerNode* && *task_state* == *state*[4]]
19.1. *accept_task* {*task_state* = *state*[5]; } → *Skip*;

If the task is rejected then its state changes to *reject* and the consumer gets a proportion of the escrow amount (line 20).

20. *RejectTask*(*vl*) = [*vl* == *consumerNode* && *task_state* == *state*[4]]
20.1. *reject_task* {*task_state* = *state*[6]; *blockchain*.*RemoveEscrowAmount*(*escrowC*/*numShares*); *consumerNode*.*AddDeposit*(*escrowC*/*numShares*); } → *Skip*;

When the task is accepted by the provider then its state is changed to *finished* (line 21.1). The provider gets the task payment and the number of activities are updated.

21. *FinishTask*(*vl*) = [*vl* == *providerNode* && *task_state* == *state*[5]]
21.1. *task_finish* {*task_state* = *state*[7]; *blockchain*.*RemoveEscrowAmount*(*escrowC*/*numShares*); *providerNode*.*AddDeposit*(*escrowC*/*numShares*); *providerNode*.*UpdateNumActivities*(1) } → *Skip*;

D. SELECTION

The consensus protocol starts with the selection of leader and validators (line 22). There should exist some management and validator nodes in the network to execute the selection, and their selection executes in parallel to save time.

22. *Selection*() = [*managementNodes*.*GetLength*() > 0 && *validatorNodes*.*GetLength*() > 0] (*LeaderSelection*() || *SelectValidator*(*K*));

1) LEADER SELECTION

The leader selection algorithm is executed by each node in parallel in a distributed environment. It is assumed that all the nodes should have exactly the same inputs to achieve synchronous agreement [63]. It is ensured using a flooding algorithm in [63]. Otherwise, disagreement may occur. Given the same input, each node chooses exactly the same leader. In case of a tie among the nodes of the same trust values, an approach similar to the work presented in [62] is used in this work to avoid this situation. Following this approach, if any two management nodes have the same trust values then a management node with the smaller hash value of identifier and trust is selected.

The process of selection of a trusted leader is specified as *LeaderSelection* (line 23). Firstly, a temporary leader is selected and its participation value is incremented (lines 23.1-23.4). The participation value of all other management nodes involved in leader selection is incremented. The deposit ratio, activity rate, and missed rate of all management nodes are compared, and the node having the highest trust value is selected as a leader (lines 23.5-23.7). In case, if the management nodes have the same trust values, then the above-described approach is used to select one leader for the current round (lines 23.8-23.11). The selection value of the leader is incremented (line 23.12). The selected leader submits the deposit in the blockchain (lines 23.13-23.16).

23. *LeaderSelection*() = *leaderSelection*{
23.1. *var index* = 0;
23.2. *mn*1 = *managementNodes*.*GetNode*(*index*);
23.3. *tempLeader* = *mn*1;
23.4. *tempLeader*.*UpdateNumParticipated*(1);
23.5. *while*(*index* < *managementNodes*.*GetLength*()−1) {
*mn*2 = *managementNodes*.*GetNode*(*index* + 1);
*mn*2.*UpdateNumParticipated*(1);
*var depositRatio*1 = *tempLeader*.*GetDeposit*()/ *managementNodes*.*GetTotalDeposit*();
*var activityRate*1 = *tempLeader*.*GetNumActivities*() / *tempLeader*.*GetNumSelected*();
*var missedRate*1 = *tempLeader*.*GetNumMisses*()/ *tempLeader*.*GetNumParticipated*();
*var depositRatio*2 = *mn*2.*GetDeposit*()/ *managementNodes*.*GetTotalDeposit*();
*var activityRate*2 = *mn*2.*GetNumActivities*()/ *mn*2.*GetNumSelected*();
*var missedRate*2 = *mn*2.*GetNumMisses*()/ *mn*2.*GetNumParticipated*();
23.6. *if* (((*depositRatio*1 + *activityRate*1 + *missedRate*1)/3) > ((*depositRatio*2 + *activityRate*2 + *missedRate*2)/3)){ *index* = *index* + 1; }
23.7. *else if* (((*depositRatio*1 + *activityRate*1 + *missedRate*1)/3) < ((*depositRatio*2 + *activityRate*2 + *missedRate*2)/3)){ *tempLeader* = *mn*2; *index* = *index* + 1; } }

23.8. *else if* (((*depositRatio*1 + *activityRate*1 + *missedRate*1)/3) == ((*depositRatio*2 + *activityRate*2 + *missedRate*2)/3)){
if ((*tempLeader*.*computeHash*(*tempLeader*.*GetMId*(), (*depositRatio*1 + *activityRate*1 + *missedRate*1)/3)) > (*mn*2.*computeHash*(*mn*2.*GetMId*(), (*depositRatio*2 + *activityRate*2 + *missedRate*2)/3))) { *tempLeader* = *mn*2;
index = *index* + 1; }
else { *index* = *index* + 1; } } }
23.9. *selectedLeader* = *tempLeader*;
23.10. *leaderSelected* = 1;
23.11. *selectedLeader*.*SetRound*(*rd*);
23.12. *selectedLeader*.*UpdateNumSelected*(1);
23.13. *escrowL* = *selectedLeader*.
23.14. *SubmitDeposit*(*selectedLeader*.*GetDeposit*());
23.15. *blockchain*.*SetEscrowAmount*(*escrowL*);
23.16. *selectedLeader*.*SetDeposit*(0); } → *Skip*;

2) VALIDATORS SELECTION

The trusted validators selection process is specified as *SelectValidator* (line 24). The deposit ratio and activity rate of all validator nodes are compared and the first k nodes having the highest trust values are selected as trusted validators (lines 24.1-24.6). The selection value of the trusted validator is incremented (lines 24.7-24.8). The selected trusted validators submit the deposit to the blockch-ain in an escrow account (lines 24.9-24.11). The record of selected trusted validators is maintained (lines 24.12-24.13).

24. *SelectValidator*(*k*) = *selectkval*{
24.1. *var index*1 = 0;
24.2. *var index*2 = 1;
24.3. *while*(*index*1 < *validatorNodes*.*GetLength*() − 1 && *index*1 < *k*) {
24.4. *while*(*index*2 < *validatorNodes*.*GetLength*()) {
*vn*1 = *validatorNodes*.*GetNode*(*index*1);
*vn*2 = *validatorNodes*.*GetNode*(*index*2);
*var depositRatio*1 = *vn*1.*GetDeposit*()/ *validatorNodes*.*GetTotalDeposit*();
*var activityRate*1 = *vn*1.*GetNumActivities*()/ *vn*1.*GetNumSelected*();
*var depositRatio*2 = *vn*2.*GetDeposit*()/ *validatorNodes*.*GetTotalDeposit*();
*var activityRate*2 = *vn*2.*GetNumActivities*()/ *vn*2.*GetNumSelected*();
24.5. *if* (((*depositRatio*1 + *activityRate*1)/2) >= ((*depositRatio*2 + *activityRate*2)/2)){
validatorNodes.*SetNode*(*index*1, *vn*1);
validatorNodes.*SetNode*(*index*2, *vn*2);
*index*2 + +; }
24.6. *else* { *validatorNodes*.*SetNode*(*index*1, *vn*2);
validatorNodes.*SetNode*(*index*2, *vn*1);
*index*2 + +; } }
24.7. *var tn* = *validatorNodes*.*GetNode*(*index*1);
24.8. *tn*.*UpdateNumSelected*(1);
24.9. *escrowV* = *tn*.*SubmitDeposit*(*tn*.*GetDeposit*());
24.10. *blockchain*.*SetEscrowAmount*(*escrowV*);
24.11. *tn*.*SetDeposit*(0);
24.12. *trustedValidatorNodes*.*AddNode*(*tn*);
24.13. *index*1 + +; } *valSelected* = 1; } → *Skip*;

E. VOTE AND BROADCAST TRANSACTIONS

After the process of selection, the leader has to prepare the block and for this purpose, the transactions that receive maximum votes from validators are required. The procedure *Vote ForTransactions* is specified at line 25. This process takes a number of transactions, and a validator as input. The number of transactions to vote for should be greater than zero and less than or equal to the maximum number of transactions (line 25.1). Otherwise, no transaction is available to vote for (line 25.2). Before the voting process, a leader and validators must be selected, otherwise, no transaction can be selected for the voting process (line 25.1).

25. *VoteForTransactions*(*t*, *vl*) =
25.1. [*t* > 0 && *t* <= *maxTransactions*]
([*leaderSelected* == 1 && *valSelected* == 1]
selection_ end → *voteTrans*(*t*, *vl*) □
[*leaderSelected* == 0 || *valSelected* == 0]
no_ transaction_ selected → *Skip*) □
25.2. [*t* < 0] *no_ transaction_ to_ vote* → *Skip*;

The voting process is described in procedure *voteTrans* (line 26). It is a sequence of three processes. In the first process, every trusted validator selects t transactions from pending transactions to vote for and the transactions proposal of every trusted validator is prepared (lines 26.2-26.3). In the second process, *VoteToTransaction_a*, the transactions proposals are validated and each trusted validator vote for it (lines 27-30). The third process describes the broadcasting and receiving of votes among validator nodes for transaction proposals (lines 26.4-26.7). A node will not broadcast or accept a block if the vote flag is zero. In this process, different communications channels are used, hence, the definition differs slightly for every node but the logical behavior remains the same.

26. *voteTrans*(*t*, *vl*) = *vote*.*t* {
26.1. *var index*1 = 0;
26.2. *while*(*index*1 < *trustedValidatorNodes*.*GetLength*() − 1) {
tvn = *trustedValidatorNodes*.*GetNode*(*index*1);
*index*1 + +; *var index*2 = 0;
26.3. *while*(*index*2 < *pendingTransactions*.*GetLength*() − 1 && *index*2 < *t* && *vl* == *tvn*.*GetVId*()) {
*tran*1 = *pendingTransactions*.*Get*(*index*2);
tTransactions.*Set*(*index*2, *tran*1); *index*2 + +; }
transactionsProposals.*Set*(*index*1,
new TransactionProposal(*tTransactions*,
new TransactionSignature(*index*2))); } } →
26.4. ([*vl* == 0] *VoteToTransaction_* 0(); (
([*voteflag*0 == 1] (*ch*01!*tempTVote*0 → *Skip* ||
*ch*02!*tempTVote*0 → *Skip* || *ch*03!*tempTVote*0 → *Skip*) □ [*voteflag*0 == 0] *Skip*) ||
([*voteflag*1 == 1] *ch*10?*y* { *tvotes*0.*Add*(*y*) } →
Skip □[*voteflag*1 == 0] *Skip*) ||
([*voteflag*2 == 1] *ch*20?*y* { *tvotes*0.*Add*(*y*) } →
Skip □ [*voteflag*2 == 0] *Skip*) ||
([*voteflag*3 == 1] *ch*30?*y* { *tvotes*0.*Add*(*y*) } →
Skip □[*voteflag*3 == 0] *Skip*)) □
26.5. [*vl* == 1] *VoteToTransaction_* 1(); . . .
26.6. [*vl* == 2] *VoteToTransaction_* 2(); . . .
26.7. [*vl* == 3] *VoteToTransaction_* 3(); . . .

In every procedure of *VoteToTransaction_a* (lines 27-30), the transaction proposal is validated and voted accordingly. If the voting behavior of a node is honest then it votes for the transactions honestly, otherwise, the node having malicious

intent does not vote for the transactions. It is mentioned that all transactions have similar behavior.

27. *VoteToTransaction_ 0() =*
validateTProposal.0 {
tempTProposal0 = transactionsProposals.Get(0);
tempProposedTransaction0 = tempTProposal0.GetTransactions();
if (voteBehaviour[0] == Honest_ Vote) {
tempTVote0 = new TransactionVote(tempProposedTransaction0, tsignature0);
tempTVote0.updateVotes(1); }
else if (voteBehaviour[0] == No_ Vote) {
voteflag0 = 0} tvotes0.Add(tempTVote0); } → Skip;
28. *VoteToTransaction_ 1() = ...*
29. *VoteToTransaction_ 2() = ...*
30. *VoteToTransaction_ 3() = ...*

In order to get the transactions with maximum votes, the process *TransWithMaxVotes* is specified (line 31). It requires t number of transactions as input. All the transactions that gain votes are checked and the first t transactions that receive maximum votes are recorded.

31. *TransWithMaxVotes(t) = trans {*
31.1. *while(index1 < votedTransactions.GetLength() − 1 && index1 < t) {*
31.2. *while(index2 < votedTransactions.GetLength()) {*
tran1 = votedTransactions.Get(index1);
tran2 = votedTransactions.Get(index2);
31.3. *if (tran1.getVotes() >= tran2.getVotes()){*
votedTransactions.Set(index1, tran1);
votedTransactions.Set(index2, tran2);
index2++; }
31.4. *else { votedTransactions.Set(index1, tran2);*
votedTransactions.Set(index2, tran1);
index2++; }
31.5. *maxVotedTransactions.Set(index1, tran1);*
31.6. *index1++; } } } → Skip;*

F. PROPOSE AND BROADCAST BLOCK

After getting transactions that gain maximum votes, the block can be proposed. The block propose and broadcast procedure is described in line 32. If the leader is selected for the round, the block b can be proposed (line 32.1). Firstly, an empty block is acquired from the locked blocks as the proposed block. The peek block of the blockchain is made the previous block of the current block to be proposed. The block is then proposed having information about the block index, leader of the block, previous block hash, and maximum voted transactions and is added in blocks proposals. If a node is the leader of the current round then it broadcasts the block to validators, otherwise, it receives the proposed block (lines 32.2-32.6). The same procedure is repeated for other blocks.

32. *ProposeAndBroadcast(b) =*
32.1. *[selectedLeader.GetRound() == b] proposeBlock.b { proposedBlock = lockedBlocks.Get(b);*
if (proposedBlock.GetBlockHash() == initialHash) { prevBlock = blockchain.GetPeekBlock();
proposedBlock = new Block(b, selectedLeader, prevBlock.GetBlockHash(), maxVotedTransactions); }
blocksProposals.SetProposal(b, new BlockProposal (proposedBlock, new BlockSignature(b))) } →
32.2. *(([b == 0] (ch01!blocksProposals.GetProposal(b) → Skip ‖ ch02!blocksProposals.GetProposal(b) → Skip ‖ ch03!blocksProposals.GetProposal(b) → Skip) □[b == 1]...[b == 2]...[b == 3]...*
32.6. *[b == 0] (ch10?y → {blocksProposals.SetProposal(b, y)} → Skip □ ch20?y → {blocksProposals.SetProposal(b, y)} → Skip □ ch30?y → {blocksProposals.SetProposal(b, y)} → Skip)□[b == 1]...[b == 2]...[b == 3]...*

After receiving the block, the validators vote for the block. The block voting process *VoteToBlock* is specified at line 33. It takes the block index b as input. It is a sequence of two processes and it is preceded by an event *propose_end* to ensure that the propose phase has ended (line 33.1). The first process *VoteToBlock_ a*() validates the block proposal and accordingly votes for the block (lines 34-37). In the second process, the votes are broadcast and received among the validator nodes (line 33.1). If the vote flag is zero, a node will not broadcast or accept the vote for the block. The same process is followed for all the blocks (lines 33.2-33.4).

33. *VoteToBlock(b) =*
33.1. *[b == 0] propose_ end → VoteToBlock_ 0(); (([voteflag0 == 1] (ch01!tempBVote0 → Skip ‖ ch02!tempBVote0 → Skip ‖ ch03!tempBVote0 → Skip) □ [voteflag0 == 0] Skip) ‖ ([voteflag1 == 1] ch10?y { bvotes0.Add(y) } → Skip □ [voteflag1 == 0] Skip) ‖ ([voteflag2 == 1] ch20?y { bvotes0.Add(y) } → Skip □ [voteflag2 == 0] Skip) ‖ ([voteflag3 == 1] ch30?y { bvotes0.Add(y) } → Skip □ [voteflag3 == 0] Skip)) □*
33.2. *[b == 1] propose_ end → VoteToBlock_ 1(); ...*
33.3. *[b == 2] propose_ end → VoteToBlock_ 2(); ...*
33.4. *[b == 3] propose_ end → VoteToBlock_ 3(); ...*

In every process of *VoteToBlock_a* (lines 34-37), the malicious behavior is introduced. Firstly, the block proposal is validated and then voted on accordingly. If a node is honest then it votes for the block honestly, otherwise, does not vote.

34. *VoteToBlock_ 0() =*
34.1. *validateProposal.0 {*
tempProposal0 = blocksProposals.GetProposal(0);
tempProposedBlock0 = tempProposal0.GetBlock();
var invalidBlock = blockchain0.ContainsBlock(tempProposedBlock0);
34.2 *if (invalidBlock) {tempProposedBlock0 = new Block(); }*
34.3. *if (voteBehaviour[0] == Honest_ Vote) {*
tempBVote0 = new BlockVote(tempProposedBlock0.GetBlockHash(), bsignature0);
tempBVote0.updateVotes(1); }
34.4. *else if (voteBehaviour[0] == No_ Vote) {voteflag0 = 0} bvotes0.Add(tempBVote0); } → Skip;*
35. *VoteToBlock_ 1() = ...*
36. *VoteToBlock_ 2() = ...*
37. *VoteToBlock_ 3() = ...*

G. BLOCKCHAIN VALIDITY

After receiving enough votes, the block can be added to the blockchain. Before adding the block, the validity of the blockchain is checked, specified as *isValidBlockchain* (line 38). The validity of the blockchain is checked by comparing the recorded and computed hash for every current and previous block (lines 38.1-38.5).

38. $isValidBlockchain() = validBlockchain\{$
38.1. $var\ index = blockchain.GetHeight();$
38.2. $while\ (index <= blockchain.GetHeight()\ \&\&\ index >= 2)\{$
$curBlock = blockchain.GetBlock(index);$
$prevBlock = blockchain.GetBlock(index - 1);$
38.3. $if\ (curBlock.GetBlockHash() != curBlock.computeHash())\{ v = Not\_Valid; \}$
38.4. $if\ (curBlock.GetPrevHash() != prevBlock.GetBlockHash())\{ v = Not\_Valid; \}$
38.5. $else\ if\ (curBlock.GetBlockHash() == curBlock.computeHash()\ \&\&\ curBlock.GetPrevHash() == prevBlock.GetBlockHash())\{$
$v = valid;\ index--;\ \}\ \}\ \rightarrow\ Skip;$

H. ADDING BLOCK IN BLOCKCHAIN

The process of block addition is specified as *AddBlock* (line 39). For every block, firstly it is ensured that the blockchain is valid (line 39.1). If the block receives votes from a majority of the validators then it is added to the blockchain. Same process is repeated for every block (lines 39.2-39.4).

39. $AddBlock(b) =$
39.1. $[b == 0\ \&\&\ v == valid]\ addtoBChain.b\ \{$
$tempProposedBlock0 = bvotes0.GetBlockWithMajorityVotes(MajorityValidators);$
$if\ (tempProposedBlock0.GetBlockHash() != initialHash\ \&\&\ untrustedLeader == false)\ \{$
$blockchain0.AddBlock(tempProposedBlock0)\ \}\ \}\ \rightarrow\ Skip\ \Box$
39.2. $[b == 1\ \&\&\ v == valid] \ldots$
39.3. $[b == 2\ \&\&\ v == valid] \ldots$
39.4. $[b == 3\ \&\&\ v == valid] \ldots$

I. UPDATE TRUST VALUES AND DEPOSIT

At the end of the consensus protocol, trust values are updated (line 40). If the block is successfully added to blockchain then the leader's number of activities is incremented, the leader gets the block generation reward and the escrow amount is returned (lines 40.1-40.2, 41-43). Similarly, the number of misses of participating management nodes are updated, the validators number of activities are incremented, they receive the block validation reward and their escrow deposit is returned (lines 40.2, 44-47). If the proposed block is not successfully added to the blockchain then the leader and validators lose trust (lines 40.3, 48-49). The leader deposit, the number of activities, and misses are set to zero. Similarly, the validators' deposit and number of activities are set to zero.

40. $UpdateTrust() =$
40.1. $[blockchain0.ContainsBlock(tempProposedBlock0)\ \|\ blockchain1.ContainsBlock(tempProposedBlock1)\ \|\ blockchain2.ContainsBlock(tempProposedBlock2)\ \|\ blockchain3.ContainsBlock(tempProposedBlock3)]$
40.2. $(UpdateLNumActivities(1);\ GetBlockGReward(escrowC/numShares);\ LDepositReturned(escrowL);\ UpdatePMNumMisses(1);\ UpdateVNumActivities(1);\ GetBlockVReward();\ TVDepositReturned();\ Skip)\Box$
40.3. $[!(blockchain0.ContainsBlock(tempProposedBlock0))\ \|!(blockchain1.ContainsBlock(tempProposedBlock1))\ \|!(blockchain2.ContainsBlock(tempProposedBlock2))\ \|!(blockchain3.ContainsBlock(tempProposedBlock3))\ ]\ maliciousLeader \rightarrow LLoseTrustAndDeposit()\ \Box\ maliciousValidators \rightarrow VLoseTrustAndDeposit();$

The *UpdateLNumActivities* process increases the number of activities of the leader by input l (line 41).

41. $UpdateLNumActivities(l) = update\ \{selectedLeader.UpdateNumActivities(l)\}\ \rightarrow\ Skip;$

In *GetBlockGReward*, the block reward is removed from the blockchain and added in the account of leader (line 42).

42. $GetBlockGReward(deposit) = getdep\ \{blockchain.RemoveEscrowAmount(deposit);\ selectedLeader.AddDeposit(deposit)\}\ \rightarrow\ Skip;$

The *LDepositReturned* process specifies that the leader's escrowed deposit is added in its account (line 43).

43. $LDepositReturned(deposit) = getdep\ \{blockchain.RemoveEscrowAmount(deposit);\ selectedLeader.AddDeposit(deposit)\}\ \rightarrow\ Skip;$

In *UpdatePMNumMisses* process, it is specified that if a management node is not a leader then its number of misses are updated (line 44).

44. $UpdatePMNumMisses(m) = update\ \{ while(index < managementNodes.GetLength() - 1)\{ mn1 = managementNodes.GetNode(index);\ if\ (mn1 != selectedLeader\ \&\&\ mn1.GetNumMisses() < maxNumMisses)\{ mn1.UpdateNumMisses(m); \}\ index++; \}\ \}\ \rightarrow\ Skip;$

The *UpdateVNumActivities* process describes that the number of activities of all the trusted validators are incremented (line 45).

45. $UpdateVNumActivities(a) = update\ \{ while(index < trustedValidatorNodes.GetLength() - 1)\ \{ tvn = trustedValidatorNodes.GetNode(index);\ tvn.UpdateNumActivities(a);\ index++; \}\ \}\ \rightarrow\ Skip;$

In *GetBlockVReward*, it is described that the validators get the block validation reward from the deposit submitted by the consumer (line 46).

46. $GetBlockVReward() = vreward\ \{ while(index < trustedValidatorNodes.GetLength() - 1)\ \{ tvn = trustedValidatorNodes.GetNode(index);\ blockchain.RemoveEscrowAmount((escrowC/numShares)/trustedValidatorNodes.GetLength());\ tvn.AddDeposit((escrowC/numShares)/trustedValidatorNodes.GetLength());\ index++; \}\ \}\ \rightarrow\ Skip;$

The *TVDepositReturned* process defines that the escrowed deposit of the validators is returned (line 47).

47. $TVDepositReturned() = tvdep\ \{ while(index < trustedValidatorNodes.GetLength() - 1)\ \{ tvn = trustedValidatorNodes.GetNode(index);\ escrowV = tvn.GetEscrow();\ blockchain.RemoveEscrowAmount(escrowV);\ tvn.AddDeposit(escrowV);\ index++; \}\}\ \rightarrow\ Skip;$

The *LLoseTrustAndDeposit* defines that the leader's number of activities, misses and deposit are set to zero (line 48).

48. $LLoseTrustAndDeposit() = ltrust$ {
$selectedLeader.SetNumActivities(0)$;
$selectedLeader.SetNumMisses(0)$;
$selectedLeader.SetDeposit(0)$; } $\rightarrow$ $Skip$;

The validators' number of activities and deposit are set to zero in *VLoseTrustAndDeposit* process (line 49).

49. $VLoseTrustAndDeposit() = vltrust$ {
$while(index < trustedValidatorNodes.GetLength() - 1)$ {
$tvn = trustedValidatorNodes.GetNode(index)$;
$tvn.SetNumActivities(0)$;
$tvn.SetDeposit(0)$; $index++$; } } $\rightarrow$ $Skip$;

J. NEXT ROUND

In order to move the consensus protocol to the next round, the *nextRound* process is specified (line 50). If the current round is less than the maximum number of rounds then the current round is incremented. All the necessary variables are cleared and the consensus protocol starts again.

50. $NextRound() =$
[$rd < maxRounds$] $nextround$ { $rd++$;
$lockedBlocks.Clear()$;
$bvotes0.Clear()$; $bvotes1.Clear()$; $bvotes2.Clear()$;
$bvotes3.Clear()$;
$tvotes0.Clear()$; $tvotes1.Clear()$; $tvotes2.Clear()$;
$tvotes3.Clear()$;
$managementNodes.Nullify(selectedLeader)$;
$validatorNodes.Nullify(trustedValidatorNodes)$;
$blockchain.Nullify(proposedBlock)$;
} $\rightarrow$ $Consensus\_Protocol() \square$
[$rd >= maxRounds$] $Skip$;

VI. RESULTS AND ANALYSIS

We have verified our formal model of the proposed consensus protocol under normal and byzantine environments using the PAT model checker. We have defined five variations of our consensus protocol. The first two variations are defined with different numbers of malicious nodes with a threat factor of no vote (lines 51-52). The next two variations are specified with different numbers of malicious nodes but the voting behavior is defined as honest (lines 53-54). The fifth variation of consensus protocol simulates the behavior of an untrusted leader (lines 55-56). After the selection process, an untrusted leader is introduced which can try to add a block in the blockchain.

51. $Consensus\_Protocol\_With\_Minority\_Malicious\_No\_Vote() = IntroduceMaliciousNodes(Minority\_Nodes, No\_Vote)$; $Consensus\_Protocol()$;
52. $Consensus\_Protocol\_With\_Majority\_Malicious\_No\_Vote() = IntroduceMaliciousNodes(Majority\_Nodes, No\_Vote)$; $Consensus\_Protocol()$;
53. $Consensus\_Protocol\_With\_Minority\_Malicious\_Honest\_Vote() = IntroduceMaliciousNodes(Minority\_Nodes, Honest\_Vote)$; $Consensus\_Protocol()$;
54. $Consensus\_Protocol\_With\_Majority\_Malicious\_Honest\_Vote() = IntroduceMaliciousNodes(Majority\_Nodes, Honest\_Vote)$; $Consensus\_Protocol()$;
55. $Consensus\_Protocol\_With\_Untrusted\_Leader() = (\| i : \{0..PNodes - 1\}@(NodeInitialization()$;
$PerformTransactions()$; $Selection()$; $IntroduceUntrustedLeader()$; $VoteForTransactions(T, i)$;
$ProposeAndBroadcast(i)$; $VoteToBlock(i)$;
$isValidBlockchain()$; $AddBlock(i)$;
$UpdateTrustAndDeposit()))$; $NextRound()$;
56. $IntroduceUntrustedLeader() =$ {
$selectedLeader = untrustedLeaderNode$;
$untrustedLeader = true$; } $\rightarrow$ $Skip$;

LTL is used to specify the desired properties. The properties are represented as assertions in Table 3. We have verified general properties, as well as security properties, under normal and byzantine environments. The verification of properties under a byzantine environment shows that the proposed model is fault tolerant.

A. DEADLOCK-FREE

A deadlock-free formal model is described as there does not occur a situation where a node has to wait for another node for an activity. The PAT model checker is used to query deadlock in the formal model. We have specified assertions and verified that the consensus protocol is deadlock-free in both normal and byzantine environments as shown in Table 3. The assertion $A1$ for the normal environment is specified below and for the byzantine environment, the assertions $A2$-$A6$ are presented in Table 3.

$A1$. $\#assert\ Consensus\_Protocol()\ deadlockfree$;

B. CONSENSUS

Consensus is defined using *#define* as every node of the network must agree on the same block and add that block to its copy of the blockchain. It is specified as the peek block of each node's chain should be the same.

$\#define\ Consensus\ (!blockchain0.IsEmpty()\ \&\&$
$blockchain0.GetPeekBlock() == blockchain1.GetPeekBlock()\ \&\&$
$blockchain1.GetPeekBlock() == blockchain2.GetPeekBlock()\ \&\&$
$blockchain2.GetPeekBlock() == blockchain3.GetPeekBlock())$;

LTL formulas are specified as assertions to verify that the formal model eventually reaches consensus or not in normal and byzantine environments. The assertion $A7$ for reaching consensus in the normal environment is specified below and for the byzantine environment, $A8$-$A12$ are represented in Table 3. The symbol $\models$ is read as satisfies. It can be noted in Table 3 that the model does not reach consensus when the majority of nodes are malicious and they do not vote ($A9$ in Table 3). It is also observed that when an untrusted leader is added in the network then consensus cannot be reached ($A12$ in Table 3). This does not show that the consensus protocol is vulnerable but it is preferred that network nodes add no block in their ledger rather than adding a malicious block.

$A7$. $\#assert\ Consensus\_Protocol() \models \Diamond\ Consensus$;

C. NO BLOCKCHAIN FORK

No blockchain fork can be assured when all nodes record the same proposed block in their blockchain. All nodes have a same view about the blockchain state.

$\#define\ No\_Blockchain\_Fork\ (!blockchain0.IsEmpty()\ \&\&$
$proposedBlock == blockchain0.GetPeekBlock()\ \&\&$

TABLE 3. Results for verification of security and trust properties against the formal model.

Assertions	Deadlock-free	⊨ ◊ Consensus	⊨ ◊ No_Blockchain_ Fork	⊨ ◊ No_selfish_ mining	⊨ ◊ No_invalid_ block_insertion	⊨ ◊ Safety	⊨ ◊ Fault_ Tolerance	⊨ ◊ Trusted_ leader	⊨ ◊ Trusted_ validators
#assert Consensus_ Protocol()	A1: Yes	A7: Yes	A13: Yes	A19: Yes	A25: Yes	A31: Yes	A37: Yes	A43: Yes	A48: Yes
#assert Consensus_ Protocol_ With_ Minority_ Malicious_ No_ Vote()	A2: Yes	A8: Yes	A14: Yes	A20: Yes	A26: Yes	A32: Yes	A38: Yes	A44: Yes	A49: Yes
#assert Consensus_ Protocol_ With_ Majority_ Malicious_ No_ Vote()	A3: Yes	A9: No	A15:No	A21: No	A27: Yes	A33: No	A39: No	A45: Yes	A50: Yes
#assert Consensus_ Protocol_ With_ Minority_ Malicious_ Honest_ Vote()	A4: Yes	A10: Yes	A16: Yes	A22: Yes	A28: Yes	A34: Yes	A40: Yes	A46: Yes	A51: Yes
#assert Consensus_ Protocol_ With_ Majority_ Malicious_ Honest_ Vote()	A5: Yes	A11: Yes	A17: Yes	A23: Yes	A29: Yes	A35: Yes	A41: Yes	A47: Yes	A52: Yes
#assert Consensus_ Protocol_ With_ Untrusted_ Leader()	A6: Yes	A12: No	A18: No	A24: No	A30: Yes	A36: No	A42: No	–	–

Yes : Property satisfied No : Property not satisfied – : Property not analyzed

proposedBlock == *blockchain*1.*GetPeekBlock*() &&
proposedBlock == *blockchain*2.*GetPeekBlock*() &&
proposedBlock == *blockchain*3.*GetPeekBlock*());

LTL formulas are specified below to verify that eventually no blockchain fork is ensured. Table 3 shows that the model ensures that a blockchain fork cannot occur in the normal environment as specified by an LTL formula *A*13 below and when the minority of nodes are malicious or byzantine nodes vote honestly as defined in Table 3 by LTL formulas *A*14 and *A*16-*A*17. The model does not satisfy this property when the majority of nodes are malicious and they do not vote (*A*15 in Table 3), and also when an untrusted leader is introduced into the network (*A*18 in Table 3). When the property is violated, then the proposed block cannot be added to the ledger of nodes but a fork does not occur.

*A*13. #*assert Consensus_ Protocol*() ⊨
◊ *No_ Blockchain_ Fork*;

D. NO SELFISH MINING

No selfish mining is ensured when a leader of a peek block of every copy of the blockchain is the same. It shows that only the block of a single leader is added to the blockchain.

#*define No_ Selfish_ Mining* (!*blockchain*0.*IsEmpty*() &&
*blockchain*0.*GetBlockLeader*(*blockchain*0.*GetPeekBlock*())
== *blockchain*1.*GetBlockLeader*(*blockchain*1.
GetPeekBlock()) && *blockchain*1.*GetBlockLeader*(
*blockchain*1.*GetPeekBlock*()) == *blockchain*2.
GetBlockLeader(*blockchain*2.*GetPeekBlock*()) &&
*blockchain*2.*GetBlockLeader*(*blockchain*2.*GetPeekBlock*())
== *blockchain*3.*GetBlockLeader*(*blockchain*3.
GetPeekBlock()));

LTL formulas are described to verify that there is no selfish mining in the protocol. The LTL formula *A*19 specified below is verified in the normal environment. In the byzantine environment, when the minority of nodes are malicious or byzantine nodes vote honestly, then the LTL formulas *A*20, *A*22-*A*23 shown in Table 3 are verified. However, when the majority of nodes become malicious and they do not vote for the block or an untrusted leader is added in the network then *A*21 and *A*24 exhibited in Table 3 are violated.

*A*19. #*assert Consensus_ Protocol*() ⊨
◊ *No_ Selfish_ Mining*;

E. NO INVALID BLOCK INSERTION

It is defined as in each round of consensus protocol there should not be any invalid (duplicate) block submitted by malicious leaders in the copy of blockchain of a node.

#*define No_ Invalid_ Block_ Insertion* (*rd* > 0 &&
!*blockchain*0.*ContainsDuplicateBlocks*() &&
!*blockchain*1.*ContainsDuplicateBlocks*() &&
!*blockchain*2.*ContainsDuplicateBlocks*() &&
!*blockchain*3.*ContainsDuplicateBlocks*());

The LTL formulas are specified to describe that the protocol and its variations satisfy eventually no invalid block insertion property. *A*25 is specified below for normal conditions while *A*26-*A*30 are defined in Table 3 for byzantine conditions. Table 3 shows that all the variations of consensus protocol satisfy this property.

*A*25. #*assert Consensus_ Protocol*() ⊨
◊ *No_ Invalid_ Block_ Insertion*;

F. SAFETY

The safety property is defined as all the nodes reach the same decision. It means that all the nodes must agree on adding the same block. It can be ensured when there are no blockchain fork and no selfish mining attacks.

#*define Safety No_ Blockchain_ Fork* &&
No_ Selfish_ Mining;

The LTL formulas are described to verify that the protocol satisfies eventually safety property under normal and byzantine conditions. When the majority of the nodes become malicious and they adopt no voting behavior or an untrutsed leader is involved in the network then assertions *A*33 and *A*36 presented in Table 3 are not satisfied. The rest of the assertions *A*31 presented below and *A*34-*A*35 represented in Table 3 are verified.

*A*31. #*assert Consensus_ Protocol*() ⊨ ◊ *Safety*;

G. FAULT TOLERANCE

In each round, a common block will be accepted by all honest nodes even in the existence of some malicious nodes. This property is specified as reaching consensus. It is described below by an LTL formula *A*37 for normal conditions and LTL formulas *A*38-*A*42 are defined in Table 3 for byzantine environment. Its behavior is similar to the consensus property as shown in Table 3.

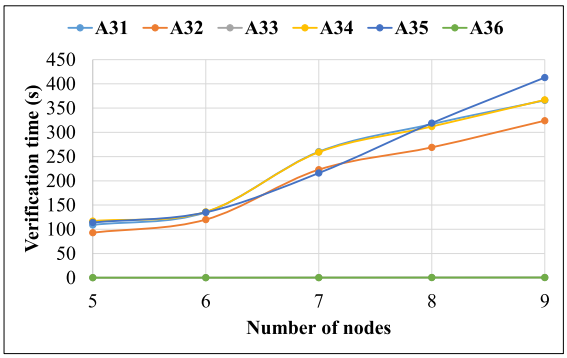

FIGURE 5. Verification time for safety.

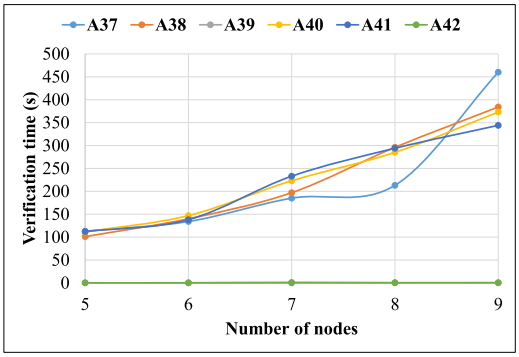

FIGURE 6. Verification time for fault tolerance.

#define Fault_ Tolerance Consensus;
*A*37. *#assert Consensus_ Protocol*() $\models \Diamond$ *Fault_ Tolerance*;

H. TRUSTED LEADER

In every round of consensus, a trusted leader is selected. The LTL formulas are specified to verify that eventually a trusted leader is selected. *A*43 is described below for the normal environment and *A*44-*A*47 for the byzantine environment are shown in Table 3. Table 3 represents that all the formulas are satisfied.

#define trustedLeader ($rd >= 0$ && *leaderSelected* == 1);
*A*43. *#assert Consensus_ Protocol*() $\models \Diamond$ *trustedLeader*;

I. TRUSTED VALIDATORS

The trusted validators are selected in every round of consensus protocol. LTL formulas are used to specify that eventually trusted validators are selected. *A*48 is defined below for the normal environment and *A*49-*A*52 are described for the byzantine environment in Table 3. This property is verified for all the variations of consensus protocol as presented in Table 3.

#define trustedValidators($rd >= 0$ && *valSelected* == 1);
*A*48. *#assert Consensus_ Protocol*() $\models \Diamond$ *trustedValidators*;

J. ANALYSIS OF THE PROTOCOL

We have performed the formal verification of the desired properties against the STBC protocol on an Intel i5-7200U 2.5 GHz CPU with 8 GB of memory, running Windows 10 and PAT version 3.5.1. The results of verification time (in seconds) of safety, fault tolerance, trusted leader, and trusted validators are presented in Figures 5, 6, 7, and 8 respectively. The number of nodes is represented on the horizontal axis and the verification time in seconds is shown on the vertical axis. The nodes involved are service consumers, service providers, blockchain management nodes, and validator nodes. When the properties are satisfied, it is analyzed that the verification time increases continuously with the increased count of nodes. In case of violation of a property, the verification time remains close to zero. For example, for the safety property, Figure 5 shows the assertions *A*31, *A*32, *A*34, and *A*35 are satisfied, therefore, the verification time increases continuously with the increased count of nodes. However, the assertions *A*33 and *A*36 are not satisfied, therefore, the verification time remains close to zero. Similarly, the verification time of fault tolerance, trusted leader, and trusted validators properties increases with the increased count of nodes. Similar behavior is observed for the verification time of deadlock, consensus, no blockchain fork, no selfish mining, and no invalid block insertion properties.

We have performed formal verification for a limited number of nodes in the network. However, the proposed

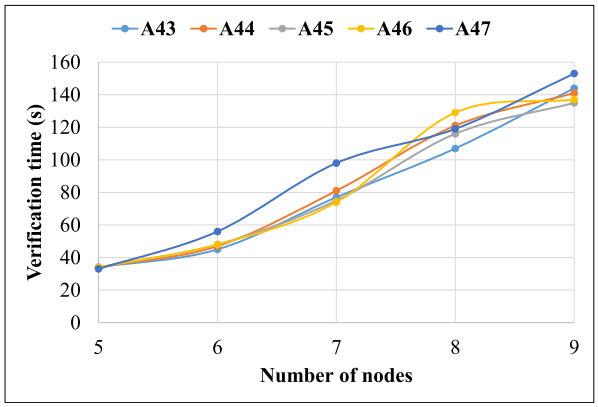

FIGURE 7. Verification time for trusted leader.

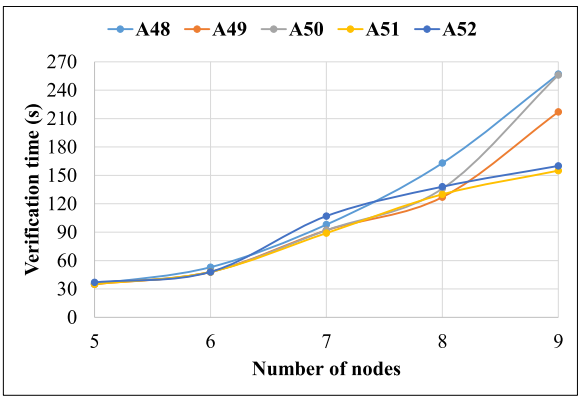

FIGURE 8. Verification time for trusted validators.

protocol itself is generic, and it can work for an arbitrary number of nodes. The noticeable point is that the verification time increases continuously with the increased count of nodes. With many nodes, an issue of state explosion may occur, which is one of the limitations of the model checking approach [33]. These limitations are outside the scope of this work. In the future, we plan to improve the efficiency of our model using different techniques, such as symbolic model checking with binary decision diagrams, partial order reduction, counterexample-guided abstraction refinement, and bounded model checking.

VII. CONCLUSION AND FUTURE WORK

Secure and Trustworthy Blockchain-based Crowdsourcing consensus protocol is a formally verified consensus protocol, which ensures safety, fault tolerance, trusted leader, and trusted validator properties. It prevents security attacks, e.g., blockchain fork, selfish mining, and invalid block insertion. The proposed consensus protocol is energy-efficient as a single leader proposes a block in each round of the protocol. The trust model ensures fairness and decentralization because the waiting nodes also get a chance to become leaders. Ensuring the correctness of a consensus protocol has significant importance because the failure of a consensus protocol can lead to disastrous consequences. Therefore, to mitigate the associated risks, we have used a formal methods-based technique, i.e., model checking, which is automatic and effective to increase the formal model's confidence of correctness. We have used CSP# for the formal specification of the consensus protocol. The security and trust properties are formally specified using LTL formulas. The PAT model checker is utilized to ensure the correctness of the formal specification against the specified properties.

At present, we have done model checking of our formal model for small number of nodes. For large number of nodes, state explosion can occur which will be addressed in future using different techniques such as symbolic model checking with binary decision diagrams, partial order reduction, counterexample-guided abstraction refinement and bounded model checking. The future work will consider failure or unavailability of nodes. Formal verification of further security properties, e.g., persistence and liveness, against the formal model will be done. We will do the performance analysis of the proposed consensus protocol using simulation techniques in addition to formal verification to get benefit of both approaches.

REFERENCES

[1] T. S. Sindlinger, "Crowdsourcing: Why the power of the crowd is driving the future of business," *Amer. J. Health-Syst. Pharmacy*, vol. 67, no. 18, pp. 1565–1566, Sep. 2010.

[2] G. Paolacci, J. Chandler, and P. G. Ipeirotis, "Running experiments on Amazon mechanical Turk," *Judgment Decis. Making*, vol. 5, no. 5, pp. 411–419, 2010.

[3] S. Hornung, D. M. Rousseau, J. Glaser, P. Angerer, and M. Weigl, "Beyond top-down and bottom-up work redesign: Customizing job content through idiosyncratic deals," *J. Organizational Behav.*, vol. 31, nos. 2–3, pp. 187–215, Feb. 2010.

[4] A. Todolí-Signes, "The end of the subordinate worker: Sharing economy, on-demand economy, crowdsourcing, uber economy and other ways of outsourcing," *SSRN Electron. J.*, pp. 5–11, Dec. 2015.

[5] M. T. Riccardi, "The power of crowdsourcing in disaster response operations," *Int. J. Disaster Risk Reduction*, vol. 20, pp. 123–128, Dec. 2016.

[6] S. Chandra, R. T. Naik, and J. Jimenez, "Crowdsourcing-based traffic simulation for smart freight mobility," *Simul. Model. Pract. Theory*, vol. 95, pp. 1–15, Sep. 2019.

[7] W. Liu, X. Wang, and W. Peng, "Secure remote multi-factor authentication scheme based on chaotic map zero-knowledge proof for crowdsourcing Internet of Things," *IEEE Access*, vol. 8, pp. 8754–8767, 2020.

[8] Downdetector. *Wikipedia Down? Current Status and Problems*. Accessed: Jan. 18, 2019. [Online]. Available: https://downdetector.com/status/wikipedia

[9] X. Zhang, G. Xue, R. Yu, D. Yang, and J. Tang, "Keep your promise: Mechanism design against free-riding and false-reporting in crowdsourcing," *IEEE Internet Things J.*, vol. 2, no. 6, pp. 562–572, Dec. 2015.

[10] D. MEYER. *Elance and Odesk Hit by DDoS*. Accessed: Jul. 28, 2020. [Online]. Available: https://gigaom.com/2014/03/18/elance-hit-by-major-ddos-attack-downing-service-for-many-freelancers/

[11] M. Isaac, K. Benner, and S. Frenkel. *Uber Hid 2016 Breach, Paying Hackers to Delete Stolen Data*. Accessed: Jul. 28, 2020. [Online]. Available: https://www.nytimes.com/2017/11/21/technology/uber-hack.htm

[12] A. Baur, "Crowdsourced formal verification: A business case analysis toward a human-centered business model," Naval Postgraduate School, Monterey, CA, USA, Tech. Rep. 0704-0188, 2015.

[13] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Tech. Rep., 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[14] F. Saleh, "Blockchain without waste: Proof-of-stake," *Rev. Financial Stud.*, vol. 34, no. 3, pp. 1156–1190, 2021.

[15] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Proc. Annu. Int. Cryptol. Conf.* Springer, 2017, pp. 357–388.

[16] X. Fu, H. Wang, and P. Shi, "A survey of blockchain consensus algorithms: Mechanism, design and applications," *Sci. China Inf. Sci.*, vol. 64, no. 2, pp. 1–15, Feb. 2021.

[17] S. Zhang and J.-H. Lee, "Analysis of the main consensus protocols of blockchain," *ICT Exp.*, vol. 6, no. 2, pp. 93–97, Jun. 2020.

[18] J. Zou, B. Ye, L. Qu, Y. Wang, M. A. Orgun, and L. Li, "A Proof-of-Trust consensus protocol for enhancing accountability in crowdsourcing services," *IEEE Trans. Services Comput.*, vol. 12, no. 3, pp. 429–445, May 2019.

[19] X. Zhu, Y. Li, L. Fang, and P. Chen, "An improved proof-of-trust consensus algorithm for credible crowdsourcing blockchain services," *IEEE Access*, vol. 8, pp. 10187–102177, 2020.

[20] W. Feng and Z. Yan, "MCS-chain: Decentralized and trustworthy mobile crowdsourcing based on blockchain," *Future Gener. Comput. Syst.*, vol. 95, pp. 649–666, Jun. 2019.

[21] S. Zhu, Z. Cai, H. Hu, Y. Li, and W. Li, "ZkCrowd: A hybrid blockchain-based crowdsourcing platform," *IEEE Trans. Ind. Informat.*, vol. 16, no. 6, pp. 4196–4205, Jun. 2020.

[22] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2014, pp. 305–319.

[23] D. Woos, J. R. Wilcox, S. Anton, Z. Tatlock, M. D. Ernst, and T. Anderson, "Planning for change in a formal verification of the raft consensus protocol," in *Proc. 5th ACM SIGPLAN Conf. Certified Programs Proofs*, Jan. 2016, pp. 154–165.

[24] K. Zheng, Y. Liu, C. Dai, Y. Duan, and X. Huang, "Model checking PBFT consensus mechanism in healthcare blockchain network," in *Proc. 9th Int. Conf. Inf. Technol. Med. Educ. (ITME)*, Oct. 2018, pp. 877–881.

[25] W. Y. M. M. Thin, N. Dong, G. Bai, and J. S. Dong, "Formal analysis of a proof-of-stake blockchain," in *Proc. 23rd Int. Conf. Eng. Complex Comput. Syst. (ICECCS)*, Dec. 2018, pp. 197–200.

[26] J. Yoo, Y. Jung, D. Shin, M. Bae, and E. Jee, "Formal modeling and verification of a federated byzantine agreement algorithm for blockchain platforms," in *Proc. IEEE Int. Workshop Blockchain Oriented Softw. Eng. (IWBOSE)*, Feb. 2019, pp. 11–21.

[27] K. Chaudhary, A. Fehnker, J. van de Pol, and M. Stoelinga, "Modeling and verification of the bitcoin protocol," 2015, *arXiv:1511.04173*.
[28] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Proc. Annu. Int. Conf. Theory Appl. Cryptogr. Techn.* Berlin, Germany: Springer, 2015, pp. 281–310.
[29] T. McGhin, K.-K. R. Choo, C. Z. Liu, and D. He, "Blockchain in healthcare applications: Research challenges and opportunities," *J. Netw. Comput. Appl.*, vol. 135, pp. 62–75, Jun. 2019.
[30] L. Cocco, A. Pinna, and M. Marchesi, "Banking on blockchain: Costs savings thanks to the blockchain technology," *Future Internet*, vol. 9, no. 3, p. 25, Jun. 2017.
[31] A. K. Kar and L. Navin, "Diffusion of blockchain in insurance industry: An analysis through the review of academic and trade literature," *Telematics Informat.*, vol. 58, May 2021, Art. no. 101532.
[32] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J.-N. Liu, Y. Xiang, and R. H. Deng, "CrowdBC: A blockchain-based decentralized framework for crowdsourcing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 6, pp. 1251–1266, Jun. 2019.
[33] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, "Model checking and the state explosion problem," in *Proc. LASER Summer School Softw. Eng.* Berlin, Germany: Springer, 2011, pp. 1–30.
[34] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," Tech. Rep., Aug. 2012, vol. 19, p. 1.
[35] P. Vasin. (2014). *Blackcoin's Proof-of-Stake Protocol v2*. [Online]. Available: https://blackcoin.co/blackcoin-pos-protocol-v2-whitepaper.pdf
[36] F. Schuh and D. Larimer. (2017). *Bitshares 2.0: General Overview*. Accessed: Jun. 2017. [Online]. Available: http://docs.bitshares.org/downloads/bitshares-general.pdf
[37] H. Sukhwani, J. M. Martinez, X. Chang, K. S. Trivedi, and A. Rindos, "Performance modeling of PBFT consensus process for permissioned blockchain network (Hyperledger Fabric)," in *Proc. IEEE 36th Symp. Reliable Distrib. Syst. (SRDS)*, Sep. 2017, pp. 253–255.
[38] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. dissertation, Univ. Guleph, Guelph, ON, Canada, 2016.
[39] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A survey of distributed consensus protocols for blockchain networks," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 1432–1465, 2nd Quart., 2020.
[40] W. Feng, Z. Yan, L. T. Yang, and Q. Zheng, "Anonymous authentication on trust in blockchain-based mobile crowdsourcing," *IEEE Internet Things J.*, early access, Aug. 24, 2020, doi: 10.1109/JIOT.2020.3018878.
[41] L. Sun, Q. Yang, X. Chen, and Z. Chen, "RC-chain: Reputation-based crowdsourcing blockchain for vehicular networks," *J. Netw. Comput. Appl.*, vol. 176, Feb. 2021, Art. no. 102956.
[42] Y. Yu, S. Liu, L. Guo, P. L. Yeoh, B. Vucetic, and Y. Li, "CrowdR-FBC: A distributed fog-blockchains for mobile crowdsourcing reputation management," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8722–8735, Sep. 2020.
[43] L. Tan, H. Xiao, X. Shang, Y. Wang, F. Ding, and W. Li, "A blockchain-based trusted service mechanism for crowdsourcing system," in *Proc. IEEE 91st Veh. Technol. Conf. (VTC-Spring)*, May 2020, pp. 1–6.
[44] M. Nguyen, Q. Bai, and J. Yu, "A blockchain-based trust model for crowd environments," in *Proc. Australas. Comput. Sci. Week Multiconf.*, Feb. 2020, pp. 1–7.
[45] S. Zhu, H. Hu, Y. Li, and W. Li, "Hybrid blockchain design for privacy preserving crowdsourcing platform," in *Proc. IEEE Int. Conf. Blockchain (Blockchain)*, Jul. 2019, pp. 26–33.
[46] N. Sukhija, E. Bautista, M. Moore, and J.-G. Sample, "Employing blockchain technology for decentralized crowdsourced data access and management," in *Proc. IEEE SmartWorld, Ubiquitous Intell. Comput., Adv. Trusted Comput., Scalable Comput. Commun., Cloud Big Data Comput., Internet People Smart City Innov. (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, Aug. 2019, pp. 268–273.
[47] Y. Lu, Q. Tang, and G. Wang, "ZebraLancer: Private and anonymous crowdsourcing system atop open blockchain," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2018, pp. 853–865.
[48] Y. Guo, H. Xie, Y. Miao, C. Wang, and X. Jia, "FedCrowd: A federated and privacy-preserving crowdsourcing platform on blockchain," *IEEE Trans. Services Comput.*, early access, Oct. 14, 2020, doi: 10.1109/TSC.2020.3031061.
[49] C. Zhang, Y. Guo, H. Du, and X. Jia, "PFcrowd: Privacy-preserving and federated crowdsourcing framework by using blockchain," in *Proc. IEEE/ACM 28th Int. Symp. Quality Service (IWQoS)*, Jun. 2020, pp. 1–10.
[50] X. Xu, Q. Liu, X. Zhang, J. Zhang, L. Qi, and W. Dou, "A blockchain-powered crowdsourcing method with privacy preservation in mobile environment," *IEEE Trans. Comput. Social Syst.*, vol. 6, no. 6, pp. 1407–1419, Dec. 2019.
[51] Y. Wu, S. Tang, B. Zhao, and Z. Peng, "BPTM: Blockchain-based privacy-preserving task matching in crowdsourcing," *IEEE Access*, vol. 7, pp. 45605–45617, 2019.
[52] M. Kadadha, R. Mizouni, S. Singh, H. Otrok, and A. Ouali, "ABCrowd an auction mechanism on blockchain for spatial crowdsourcing," *IEEE Access*, vol. 8, pp. 12745–12757, 2020.
[53] L. Gao, T. Cheng, and L. Gao, "Tswcrowd: A decentralized task-select-worker framework on blockchain for spatial crowdsourcing," *IEEE Access*, vol. 8, pp. 220682–220691, 2020.
[54] J. Wang, G. Sun, Y. Gu, and K. Liu, "ConGradetect: Blockchain-based detection of code and identity privacy vulnerabilities in crowdsourcing," *J. Syst. Archit.*, vol. 114, Mar. 2021, Art. no. 101910.
[55] S. Rasool, M. Iqbal, T. Dagiuklas, Z. Ul-Qayyum, and S. Li, "Reliable data analysis through blockchain based crowdsourcing in mobile ad-hoc cloud," *Mobile Netw. Appl.*, vol. 25, no. 1, pp. 153–163, Feb. 2020.
[56] M. Sivaram, G. Rathee, R. Rastogi, M. T. Quasim, and H. Saini, "A resilient and secure two-stage ITA and blockchain mechanism in mobile crowd sourcing," *J. Ambient Intell. Hum. Comput.*, vol. 11, pp. 1–14, Nov. 2020.
[57] S. Wang, A. F. Taha, and J. Wang, "Blockchain-assisted crowdsourced energy systems," in *Proc. IEEE Power Energy Soc. Gen. Meeting (PESGM)*, Aug. 2018, pp. 1–5.
[58] S. Wang, A. F. Taha, J. Wang, K. Kvaternik, and A. Hahn, "Energy crowdsourcing and peer-to-peer energy trading in blockchain-enabled smart grids," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 49, no. 8, pp. 1612–1623, Aug. 2019.
[59] H. Ma, E. X. Huang, and K.-Y. Lam, "Blockchain-based mechanism for fine-grained authorization in data crowdsourcing," *Future Gener. Comput. Syst.*, vol. 106, pp. 121–134, May 2020.
[60] K. Nelaturu, J. Adler, M. Merlini, R. Berryhill, N. Veira, Z. Poulos, and A. Veneris, "On public crowdsource-based mechanisms for a decentralized blockchain oracle," *IEEE Trans. Eng. Manag.*, vol. 67, no. 4, pp. 1–15, Nov. 2020.
[61] Y. Ding, Z. Chen, F. Lin, and C. Tang, "Blockchain-based credit and arbitration mechanisms in crowdsourcing," in *Proc. 3rd Int. Symp. Auto. Syst. (ISAS)*, May 2019, pp. 490–495.
[62] J. Yu, D. Kozhaya, J. Decouchant, and P. Esteves-Verissimo, "RepuCoin: Your reputation is your power," *IEEE Trans. Comput.*, vol. 68, no. 8, pp. 1225–1237, Aug. 2019.
[63] J. Aspnes, "Notes on theory of distributed systems," 2020, *arXiv:2001.04235*.

HAMRA AFZAAL received the B.S.C.S. and M.S.C.S. degrees from the Department of Computer Science, COMSATS University Islamabad (CUI), Sahiwal, Pakistan. She is currently pursuing the Ph.D. degree with Information Technology University, Lahore, Pakistan. She joined the Department of Computer Science, CUI, as a Lecturer, in 2016. Her research interests include blockchain, consensus protocols, wireless sensor and actor networks, and formal modeling and verification using formal methods.

MUHAMMAD IMRAN (Member, IEEE) received the Ph.D. degree in information technology from University Teknologi PETRONAS, Malaysia, in 2011. He has 15 years of tertiary level teaching experience in Pakistan, Malaysia, Saudi Arabia, and Australia. He is currently a Senior Lecturer with the School of Science, Engineering and Information Technology, Federation University Australia. Previously, he worked at King Saud University (KSU), Saudi Arabia, as an Assistant and Associate Professor, from 2011 to 2021. He has taught diverse courses, including computer networks, network protocols & algorithms, data science, information security, E-business, data structures, and operating systems. He is also the Founding Leader of the Wireless Networks and Security (WINS) Research Group, KSU, from 2013 to 2021. His research is financially supported by several national and international grants. He has completed a number of international collaborative research projects with reputable universities. He has published more than 300 research articles in peer-reviewed, highly-reputable international conferences (90), journals (198), editorials (15), book chapter (one), and two edited books. Many of his research articles are among the highly cited and most downloaded. He has been listed among top 100 researchers by the Thomson Reuters (Web of Science) based on the number of citations earned in the last five years in the computer science and information systems category. His research has been cited more than 9,0000 with H-index of 47, and i-10 index of 156 (Google Scholar). His research interests include mobile and wireless networks, the Internet of Things, big data analytics, cloud/edge computing, and information security. He has received a number of awards and fellowships. He served as an Editor-in-Chief for *European Alliance for Innovation (EAI) Transactions on Pervasive Health and Technology* and an Associate Editor for *IEEE Communications Magazine*. He is serving as an Associate Editor for top ranked international journals, such as IEEE Network, IEEE Future Generation Computer Systems, and IEEE Access. He served/serving as a Guest Editor for about two dozen special issues in journals, such as *IEEE Communications Magazine*, *IEEE Wireless Communications Magazine*, IEEE Future Generation Computer Systems, IEEE Access, and IEEE Computer Networks. He has been involved in about 100 peer-reviewed international conferences and workshops in various capacities, such as the chair, the co-chair, and a technical program committee member. He has been consecutively awarded with the Outstanding Associate Editor of IEEE Access, in 2018 and 2019, besides many others.

MUHAMMAD UMAR JANJUA received the Ph.D. degree from Cambridge University. He is currently an Assistant Professor with the Department of Computer Science, Information Technology University (ITU), Lahore, Pakistan, where he is also the Director of the Blockchain Research Laboratory. From 2008 to 2017, he worked at Microsoft Corporation, as a Software Engineer, with the Windows Security Research and Development Group. He has been the primary developer in preparing several mitigation packages for the entire Windows ecosystem. His research interests include static analysis, program verification and synthesis, big data security, and applied cryptography.

SARADA PRASAD GOCHHAYAT received the M.Tech. degree in signal processing from IIT Guwahati, India, in 2010, and the Ph.D. degree in communication engineering from the Indian Institute of Science, India, in 2016. From 2016 to 2018, he was a Postdoctoral Fellow with the Department of Mathematics, University of Padua, Italy. He is currently a Postdoctoral Fellow with the Virginia Modeling, Analysis and Simulation Center, Suffolk, VA, USA, and an Adjunct Faculty with Old Dominion University, Norfolk, USA. His research interests include security and privacy in distributed computing and networks, especially in the IoT, cloud computing, and blockchain.