Brief Announcement: Holistic Verification of Blockchain Consensus

Nathalie Bertrand, Vincent Gramoli, Igor Konnov, Marijana Lazić, Josef Widder, Pierre Tholoniat

To cite this version:


HAL Id: hal-03819744
https://inria.hal.science/hal-03819744
Submitted on 18 Oct 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Brief Announcement: Holistic Verification of Blockchain Consensus

Nathalie Bertrand  
nathalie.bertrand@inria.fr  
Univ Rennes, Inria, CNRS, IRISA  
Rennes, France

Vincent Gramoli  
vincen.gramoli@sydney.edu.au  
University of Sydney  
Sydney, Australia

Igor Konnov  
igor@informal.systems  
Informal Systems  
Wien, Austria

Marijana Lazić  
lazic@in.tum.de  
TU Munich  
Munich, Germany

Pierre Tholoniat  
pierre@cs.columbia.edu  
Columbia University  
New York City, USA

Josef Widder  
josef@informal.systems  
Informal Systems  
Wien, Austria

ACM Reference Format:  

1 INTRODUCTION AND PRELIMINARIES

Motivation. Today, the market capitalization of the seminal blockchain, Bitcoin, is about $803B which incentivizes malicious participants to find problematic executions that would allow them to steal financial assets. As the blockchain requires a distributed set of machines to agree on a unique block of transactions to be appended to the chain, attackers naturally try to exploit consensus vulnerabilities to double spend. As a result, formally verifying that a blockchain consensus protocol is safe and live is key to mitigate financial losses. Recent progress in mechanical proofs represent the first steps towards verifying blockchain consensus. The parameterized model checking of threshold automata (TAs) has recently proved instrumental in verifying fully asynchronous parts of consensus algorithms, like broadcast algorithms [4]. The aforementioned reduction technique cannot apply to partial synchrony: moving the message reception step to a later point in the execution might violate an assumed message delay.

Contribution. In this paper, we verify holistically the safety and liveness properties of the DBFT [2] Byzantine consensus used in the Red Belly Blockchain system [3], a scalable blockchain used in production. Our approach is holistic because it starts from the pseudocode of the distributed algorithm as typically presented in the distributed computing literature, models this pseudocode and its components into disambiguated TA, model checks the desired properties of these components expressed in LTL formulae, simplifies the TA of the consensus algorithm with these verified properties and model checks the safety and liveness of the consensus protocol. The advantage is that the formally verified algorithm matches the pseudocode and no user-defined invariants or proofs need to be checked, which drastically reduces the risks of human errors.

Model. We consider \( n \) asynchronous sequential processes \( P = \{p_1, \ldots, p_n\} \) that communicate by exchanging messages through an asynchronous reliably fully connected point-to-point network. A set of \( f \leq t < n/3 \) processes are Byzantine, the rest are correct. Distributed algorithms are communication-closed and their execution consists in an interleaving of the individual steps taken by the processes. A threshold automaton (TA) describes the behavior of a process in a distributed algorithm. Its nodes represent local states, and labeled edges are guarded rules. In order to verify the safety and liveness properties of our distributed algorithms on a multi-round TA, we express these properties in fragments of linear temporal logic (LTL). Since the algorithms are communication-closed, the verification on the multi-round TAs reduces to one-round TAs. We then use the ByMC model checker to automatically verify these fragments.

2 THE BINARY VALUE BROADCAST

Algorithm 1 Binary value broadcast for process \( p_1 \).

1. \texttt{bv-broadcast} (BV, \langle val, i \rangle);
2. \texttt{broadcast} (BV, \langle val, i \rangle);
3. \texttt{repeat}:
4. if \((\text{BV}, \langle v, * \rangle)\) received from \((t+1)\) distinct proc. (not yet re-bcast) then
5. \( \text{broadcast} (\text{BV}, \langle v, i \rangle) \)
6. if \((\text{BV}, \langle v, * \rangle)\) received from \((2t+1)\) distinct processes then
7. contestants \( \leftarrow \) contestants \( \cup \) \{v\}

Our holistic verification approach consists of decomposing a distributed algorithm into encapsulated components of pseudocode that can be modelled in TAs and verified in isolation to obtain a simplified TA that is amenable to automated verification. The binary value broadcast (bv-broadcast) [5] is a communication primitive guaranteeing that all binary values "bv-delivered" were "bv-broadcast" by a correct process. Each process starts this algorithm in one of two states, depending on its input value 0 or 1. Once a correct process receives a value from \( t+1 \) distinct processes, it broadcasts it if it did not do it already (line 4); broadcast is not
The locations of the automaton correspond to the exclusive situations for a correct process. The initial location (Alg. 1) parameterized by \( n \) mimicking the broadcast of a new message of type \( \sigma \). Processes in \( BV_{-Obligation} \), \( BV_{-Uniformity} \) and \( BV_{-Termination} \). Here, we for-

properties. We thus introduce the fairness assumption that will be crucial in the rest of this paper. In order to define it, we first define a good execution of the \( bv \)-broadcast with respect to binary value \( v \) as an execution:

**Definition 2.1** \((v\text{-good } bv\text{-broadcast})\). A \( bv \)-broadcast execution is \( v\text{-good} \) if all its correct processes \( bv\text{-deliver} v \) first, i.e., \( \square (x[C_{1-v}对自己}{0} ∧ x[C_{1-v}对自己}{0}) \).

Second, we consider an infinite sequence of \( bv \)-broadcast executions, tagged with \( r \in \mathbb{N} \). Since we replace partial synchrony by our fairness assumption, the setting is fair but asynchronous, that is, processes invoke \( bv \)-broadcast infinitely many times, but at their own relative speed. Thus, they do not all invoke the \( bv \)-broadcast tagged with the same number \( r \) at the same time. Nonetheless, every process invokes \( bv \)-broadcast infinitely many times and in the \( r\text{-th} \) invocation its behavior depends on the messages sent in the \( r\text{-th} \) invocation of other processes. Therefore, we refer to the \( r\text{-th} \) execution of \( bv \)-broadcast even though the processes invoke it at different times.

**Definition 2.2** (fairness). An infinite sequence of \( bv \)-broadcast executions is fair if there exists an \( r \) such that the \( r\text{-th} \) execution is \((r \mod 2)\)-good.

For simplicity, we use the terminology fair \( bv \)-broadcast when the infinite sequence of \( bv \)-broadcast executions is fair.

### Algorithm 2 The Byzantine consensus algorithm at process \( p_i \)

1. Global scope variable:
   \[ \text{contestants} \subseteq \{0,1\} \text{, set of binary values, initially } 0. \]

2. propose(e)
   \[ r \leftarrow 0. \]

3. repeat
   \[ \text{broadcast}(est, (est, i)) \]
   \[ \text{wait until} \ (\text{contestants} \neq \emptyset) \]
   \[ \text{broadcast}(\text{aux}, (\text{contestants}, i)) \rightarrow \text{favorites} \]
   \[ \text{wait until} \ 3c_1, \ldots, c_{n-t} \vdash \forall i \leq j \leq n-t \text{ favorites}[c_j] \neq \emptyset \]
   \[ \land (\text{qualifiers} \leftarrow \{ c_j \in \text{contestants} \mid \text{favorites}[c_j] \subseteq \text{contestants} \}) \]
   \[ \text{if} \text{ qualifiers} = \{ x \} \text{ then} \]
   \[ est \leftarrow a \]
   \[ \text{if} \ a = (r \mod 2) \text{ then} \text{ decide(a) } \]
   \[ \text{else est} \leftarrow (r \mod 2) \]
   \[ r \leftarrow r + 1 \]

### 3 Consensus Verification

In this section, we leverage our verification of \( bv \)-broadcast to simplify the verification of the TA of DBFT.

**The Byzantine consensus algorithm.** Since we replace the partial synchrony assumption by fairness, DBFT can be simplified into Algorithm 2, without coordinators and timeouts. The DBFT binary consensus invokes \( bv\text{-broadcast}(\cdot) \) at line 6 and uses a set \( \text{contestants} \) of binary values, whose scope is global, updated by the \( bv\text{-broadcast} \) (Alg. 1, line 7) and accessed by the procedure propose(\( \cdot \)) (Alg. 2, line 7).

**Modeling deterministic consensus.** Figure 2 depicts the TA obtained by modeling Algorithm 2 with the aforementioned method.
We can formally verify that DBFT (Alg. 2) solves the Byzantine consensus with the fair bv-broadcast and without partial synchrony. The full proofs can be found in the companion technical report [1]. First, we verify safety (validity and agreement) by model-checking two well-chosen superround invariants. For both values of $\nu$ we obtain:

$$\forall R \in \mathbb{N}, \forall R' \in \mathbb{N} \left( 0 \land \kappa[R, D_{\nu}] \neq 0 \Rightarrow \square \kappa[D_{\nu}, R'] = 0 \right) \quad \text{(Agree)}$$

$$\forall R \in \mathbb{N} \left( \kappa[V_{\nu}, 1] = 0 \Rightarrow \square \kappa[D_{\nu}, R] = 0 \right) \quad \text{(Valid)}$$

Second, we verify liveness (termination) by model-checking four properties with ByMC and conclude with a short proof that relies on our fairness assumption (cf. Def. 2.2):

Theorem 3.1. Assuming fairness of the bv-broadcast, Algorithm 2 terminates.

Experiments. Thanks to our modular approach, we are able to verify all properties of the Byzantine consensus automatically in about 70 seconds whereas a non-compositional approach timed out after more than 24 hours.

REFERENCES


