Specifying and checking an extension of Tendermint consensus in TLA$^+$

Jure Kukovec[*], Daniel Cason, Igor Konnov, and Josef Widder

Informal Systems, Austria and Switzerland

Introduction. The Tendermint consensus protocol [7] lies at the very heart of Cosmos, powering dozens of blockchains [2]. It is a member of the Byzantine fault-tolerant family of protocols, that is, it can achieve consensus even if a fraction of consensus participants (concretely, less than one third) behave arbitrarily and are allowed to deviate from the protocol.

Informal Systems maintains a TLA$^+$specification of the Tendermint protocol [5]. It defines a number of invariants: agreement, validity and accountability, as well as an inductive invariant for proving these properties for arbitrarily long executions with the Apalache model checker [1].

Tendermint blockchains assign a timestamp to every block, which is required by some applications. The original Tendermint computes time as follows: validators piggyback their local time readings onto precommit messages. The block proposer computes the block timestamp as the median of the timestamps in the received precommit messages for the previous block. The protocol does not make any assumptions regarding the clocks of validators, as block timestamps do not affect consensus.

In the recent extension of Tendermint, the timestamp is suggested by the block proposer. A block validator accepts the proposed block (and the timestamp), if it is received within a certain time window, determined by the synchrony parameters and the local clock of the validator. We call this version proposer-based timestamp, or PBTS for short. Accompanying the protocol change [4], we have updated the TLA$^+$ specification [6]. In this talk, we present the challenges and trade-offs encountered while attempting to verify the updated specification.

Trade-off 1: Readability vs. checking times. By design, TLA$^+$is a language meant for humans to write, and importantly, read and understand. Thus, there is no inherent notion of execution or performance; there is no such thing as a slow or inefficient specification. Of course, in practice, specifications are also

[*]Corresponding author: jure@informal.systems

inputs to automated tools. Concretely, we want to feed our specifications to the Apalache model checker. This means that, when authoring a specification, one needs to consider that the most readable, or most compact specification may be one that is slow for tools to analyze.

Trade-off 2: Modeling faults. We have implemented two mechanisms for modeling the behavior of Byzantine validators:

1. "Preloading" all messages by faulty validators in the initial state. Since faulty validators do not have to respect the protocol, they can send messages with arbitrary timestamps and payloads at any point in time. Hence, all messages by faulty processes can be modeled to have already been sent in the initial state and read by correct processes later.

2. Sending messages by faulty validators one-by-one. Just like how a correct validator can send a message (with a restricted payload) in one step, we model a similar step for faulty validators, except the payload is arbitrary.

Ultimately, the set of reachable states is the same for both approaches. However, the first approach utilizes data non-determinism, whereas the second approach utilizes control non-determinism. The first approach reaches the same state via a shorter sequence of transitions. In practice, model checking with Apalache is more efficient for shorter executions.

Additionally, to get faster user feedback, we show how to constrain the number of preloaded messages and thus reduce the model checking times. We use Apalache generators that are similar to generators in property-based testing. Intuitively, this is another instance of the small scope hypothesis; one typically needs only a handful of messages by faulty processes to find invariant violations, if they exist.

Experiments. We have checked the key properties of Tendermint: agreement, validity, and time invariant. Importantly, we considered two sets of configurations for four validators: (1) when correct validators are in a two-thirds supermajority, and (2) when there is no supermajority of correct validators. As expected, Apalache has shown that there are no invariant violations for bounded executions in configuration (1), and there is a counterexample to the properties in configuration (2). In future work, we aim to devise an inductive invariant for this specification and prove correctness for arbitrary trace length. The experimental results are available from [3].

References

[1] *Apalache model checker.* `https://apalache.informal.systems/`.

[2] *Cosmos: The Internet of Blockchains.* `https://cosmos.network/`.

[3] *PBTS model checking results.* `https://github.com/tendermint/tendermint/blob/main/pbts/spec/consensus/proposer-based-timestamp/tla/experiment_log.md`.

[4] *Proposer-Based Timestamps (PBTS).* `https://github.com/tendermint/tendermint/tree/master/spec/consensus/proposer-based-timestamp`.

[5] *TLA$^+$specification of Tendermint.* `https://github.com/tendermint/tendermint/blob/master/spec/light-client/accountability/TendermintAcc_004_draft.tla`.

[6] *TLA$^+$specification of the PBTS Tendermint variant.* `https://github.com/tendermint/tendermint/blob/main/pbts/spec/consensus/proposer-based-timestamp/tla/TendermintPBT.tla`.

[7] *The latest gossip on BFT consensus.* `https://arxiv.org/abs/1807.04938`.