Formal Modeling and Verification of a Federated Byzantine Agreement Algorithm for Blockchain Platforms

Junghun Yoo*, Youlim Jung†, Donghwan Shin‡, Minhyo Bae§ and Eunkyoung Jee†
* Department of Computer Science, University of Oxford, Oxford, UK
junghun.yoo@cs.ox.ac.uk
† School of Computing, KAIST, Daejeon, Republic of Korea
yljung@se.kaist.ac.kr ekjee@se.kaist.ac.kr
‡ SnT, University of Luxembourg, Luxembourg
donghwan.shin@uni.lu
§ BlockchainOS Inc., Seoul, Republic of Korea
minhyo.bae@blockchainos.org

***Abstract*—A blockchain is a type of distributed ledger that can record transactions between parties in a verifiable and permanent manner. Each node contains its ledger, and the contents of each ledger are maintained to be the same by a consensus algorithm. It is essential to ensure the safety and liveness of the consensus algorithms in blockchain platforms. The Stellar Consensus Protocol (SCP), which is a consensus algorithm for the Stellar cryptocurrency using the blockchain, is utilized for the federated Byzantine agreement. The quorum configuration is one of the essential factors for ensuring the safety and liveness of the SCP; however, it has been rarely studied. In this study, we model the SCP with timed automata and verify the model using a model checking technique, with the purpose of investigating and evaluating the SCP. Through the modeling and verification of the SCP, we could check whether a certain quorum configuration ensures consensus or not, before execution on an actual network. We present several abstraction techniques that help in coping with the extremely large state space of the SCP model in formal verification. The proposed modeling and verification techniques can be utilized for other consensus protocols of various blockchain platforms using the Byzantine agreement.**

***Index Terms*—consensus, federated Byzantine agreement, Stellar Consensus Protocol, blockchain, modeling, verification, UPPAAL**

I. Introduction

A blockchain is a type of distributed ledger that can record transactions between parties in a verifiable and permanent manner. Each node contains its ledger, and the contents of each ledger are maintained to be the same by a consensus algorithm. It is essential to ensure the safety and liveness of the consensus algorithms in blockchain platforms.

The Stellar Consensus Protocol (SCP) [1] is a consensus algorithm for the Stellar cryptocurrency using the blockchain. It is a construction for the federated Byzantine agreement (FBA). Nodes in FBA construct a finite number of quorums and each quorum has its own threshold for the agreement of the quorum. FBA can guarantee a successful agreement if all quorums are connected with common nodes (a Quorum Intersection) and each quorum can reach a successful agreement for itself. The SCP is claimed to provide a considerably faster block creation time compared to proof-of-work consensus algorithms such as Bitcoin. Even though the safety and liveness of the SCP have been mathematically proved [1], the SCP can only ensure safety when nodes select adequate quorum slices. The quorum configuration is an essential factor for ensuring the safety and liveness of the SCP; however, it has been rarely studied in the practical context. In this study, we model and verify the SCP using UPPAAL [2], which is a widely used real-time model checker, with the purpose of investigating and evaluating the SCP. First, we explain how we build the SCP and an environment model, which is a network circumstance in this case. Then, simulation results and formal verification results with respect to primary properties are demonstrated.

Model checking is a promising technique for automatically verifying the correctness properties of a finite-state system. However, the state explosion problem is a major obstacle that makes the industrial application of model checking difficult. As the model for the SCP consists of a tremendous number of states, appropriate abstraction should be performed on the model. We identified several factors that have high impact on the state explosion problem of the SCP model and parameterized these factors for state space reduction.

Through the modeling and verification of the SCP, we could confirm that there are no major flaws in the SCP and check whether a certain quorum configuration ensures consensus or not, before execution on an actual network.

The contributions of this work are as follows:

- We developed timed automata models for the SCP. These models enable users to easily check whether a quorum configuration ensures consensus achievement or not.
- We identified and parameterized major factors that can resolve the state explosion problem in the modeling and verification of the SCP.

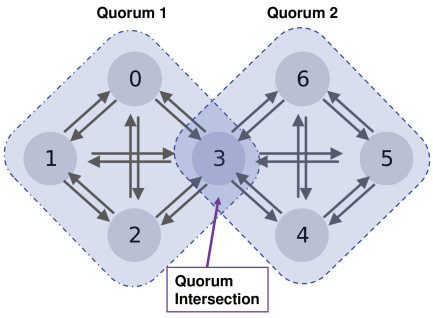


Fig. 1. Two Quorums with One Quorum Intersection

- The presented modeling and verification practices can be utilized for the modeling and verification of other protocols based on the Byzantine agreement.

The rest of this paper is structured as follows: Section II explains the SCP and UPPAAL model checker, and Section III presents the modeling of the SCP using UPPAAL. Section IV explains the verification properties, corresponding results, and state space reduction techniques. Section V discusses issues in modeling and verification of the SCP, and Section VI analyzes relevant studies. Section VII concludes this paper and presents the direction of future work.

II. Background

A. Stellar Consensus Protocol

The SCP is a consensus algorithm that is used in Stellar cryptocurrency. Mazires [1] attempted to build an application that can handle open membership. In the aspect of worldwide financial networks, he addressed the importance of constructing an adjustable network that an organization can join in or leave from freely while retaining the integrity of transaction history and suggested the FBA. An FBA system (FBAS) is composed of quorums. A node in the FBAS can select nodes to trust, and these nodes are referred to as a quorum slice. A quorum is a set of nodes that can lead to consensus, and it must contain the quorum slices of a quorum member. If there are quorums sharing some nodes, then a quorum intersection is formed based on the shared nodes. Each node counts the votes of the quorum slices to check whether the nodes in the quorum slices can reach a threshold. Checking quorum slices using threshold rather than whole quorum agreement can lead to the rapid achievement of consensus. For instance, in Fig. 1, we can notice that node 0 trusts node 1, 2, 3 and itself, and these nodes are a quorum slice of node 0. Furthermore, it is possible to gather information of quorum slices of the rest of nodes. These information of quorum slices leads that node 0, 1, 2 and 3 construct a quorum because the quorum can contain the quorum slices of node 0, 1, 2 and 3. If there is another quorum containing node 3, the shared node 3 would be a quorum intersection. When a threshold in percentile is set to 67%, a threshold of node 0 for quorum agreement would be 3 because the number of its quorum slice member is 4, and 67% of 4 is 3. Node 0 would follow the voting results of three or more nodes. In this way, nodes decide their vote using partial agreement of quorum slices, and it is possible to derive a quick consensus of intact nodes, which select their quorum slices properly and respond to all requests eventually with correct results.

Nodes maintain a ledger by storing their transactions in a slot when they agree on the transactions. In the FBAS, it is possible to achieve agreement between nodes by using federated voting through the three steps of 'vote,' 'accept,' and 'confirm.' If the number of votes of a certain value exceeds thresholds, the value proceeds to the final consensus. If nodes agree on the value, it is stored in a slot. The SCP is based on the FBAS, and it undergoes through two processes for agreement, as described in Fig. 2. First, a nomination protocol determines the value to be written to a slot. In the protocol, each node contains *voted* and *accepted* lists. *voted* consists of candidate values that can be used as nominated values. During the nomination round, a node selects the most significant node among the quorum slice in the current round, and copies the items in *voted* and *accepted* lists of the significant node to its *voted*. If a node confirms that a value in its *voted* list reaches the threshold in the *voted* and *accepted* of quorum slice members through messages, it can move the value to the *accepted* list. In this case, the threshold is referred to as the quorum threshold. On the contrary, if a value is in *accepted* for quorum slice members that are more than a certain number, this value can be added to *accepted* directly even though it is not in *voted*. In this case, the threshold is referred to as the blocking threshold. Then, if a value in the *accepted* list of a node has been accepted by members that are more than the quorum threshold, it can be confirmed as a nominated value. Once this value has been confirmed, it immediately prevents new candidates coming in from the input value pool from being voted.

The ballot protocol starts when the nomination protocol confirms a value as a nominated value. A ballot is created into a pair of this value and counter, and the three phases of PREPARE, COMMIT, and EXTERNALIZE are performed. In the PREPARE phase, federated voting is used to combine the nominated values created from the nomination protocol operating in parallel. When the combined value is confirmed, the protocol enters into the COMMIT phase, and the nomination protocol is no longer active. If the voting process can confirm the value, the process enters the EXTERNALIZE phase. In this phase, an EXTERNALIZE message is sent so that all other nodes can rapidly reach a consensus. Finally, the value confirmed in the previous phase is stored in the slot.

B. UPPAAL Model Checker

As the SCP algorithm contains timing constraints such as timeout, we model and verify it using UPPAAL. UPPAAL is a verification tool based on timed automata that was developed to verify real-time systems [2]. It was first proposed in 1995 [3], and it has been developed continuously since then. Numerous studies have used UPPAAL as a verification tool

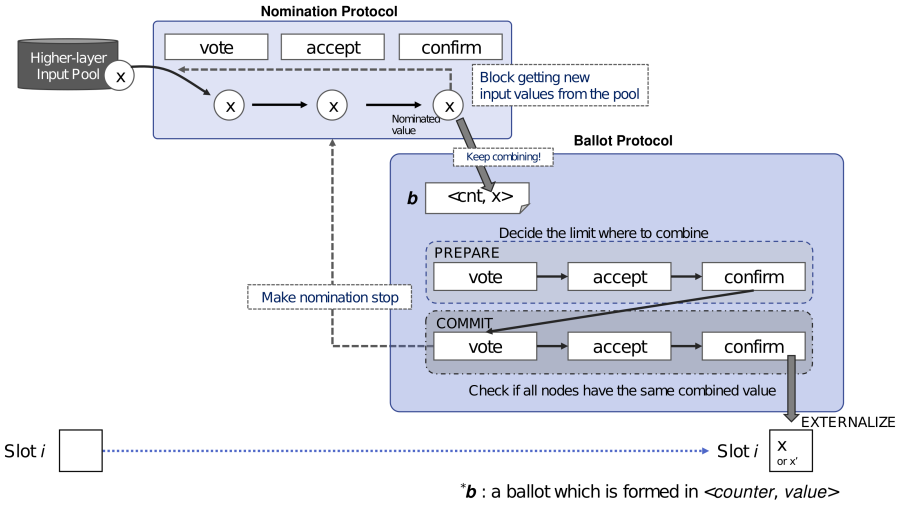


Fig. 2. Overview of Stellar Consensus Protocol

[4]–[8]. UPPAAL can be divided into three parts: an editor, a simulator, and a verifier. The system of interest can be represented as processes, which can be defined using templates in the editor. *Templates* are defined by state-based diagrams and textual declaration. The textual declaration contains the variables and functions used in the diagrams, written in C-like language. In addition to the templates, *Global declaration* defines the variables that are used throughout the system, and *System declaration* defines entire system by specifying templates to be included in the system as processes. The system can be executed manually or randomly in the simulator. Therefore, it is possible to obtain traces of the system and observe how globally declared variables change. The verifier uses the query language to represent one or more properties and is able to optionally check their satisfaction. Additionally, it provides a counterexample for a case where a certain property is not satisfied.

III. SCP Modeling

As described in Section II, the SCP consists of two sub-protocols: the nomination protocol and ballot protocol. Each protocol exchanges messages, which contain the latest states of the sender node, through an asynchronous network. The completion of one cycle of the protocol (externalizing an agreed value on all intact nodes) implies a successful agreement on a slot in our model. Each slot can be treated as a single block in the blockchain because an externalized value can be combined with multiple input values. Our model[1] consists of four templates: the nomination template, ballot template, slot manager template, and nomination observer template. The nomination observer template is for the correctness of the nomination protocol, and it is described in Section IV.

[1]The UPPAAL models presented in this paper is available at https://github.com/eunkjee/scp-uppaal-model-iwbose19.

```
// total number of nodes
const int N = 4;

// global message store for
//     nomination protocol
SCPNominationState nominationStates[N];

// global message store for
//     ballot protocol
SCPBallotState ballotStates[N];
```

Fig. 3. The Global Message Store

A. Network Model

The SCP assumes that all nodes are on the asynchronous network. The asynchronous network might experience certain issues in exchanging messages: a) failing to deliver messages (including the delivery of damaged messages), b) delivering duplicated messages, c) delivering messages out of order, and d) delaying to deliver messages. This implies that it is difficult for a node to determine whether the received messages are valid and/or in the correct order. Therefore, the SCP uses totally ordered messages to prevent these issues. The receiving nodes can identify the latest message of a specific node among the messages from that node. Furthermore, the validity of the messages is ensured by the cryptography technique (sending a hash value first and checking the contents with the hash value later) in an actual system. Hence, our model does not require the validity check for the messages.

Our network model maintains a global message store instead of individual message queues for each node to reduce the overhead caused by exchanging messages (See Fig. 3). Each node stores its latest message, i.e., its state, to the global message store by maintaining the order of the messages. The

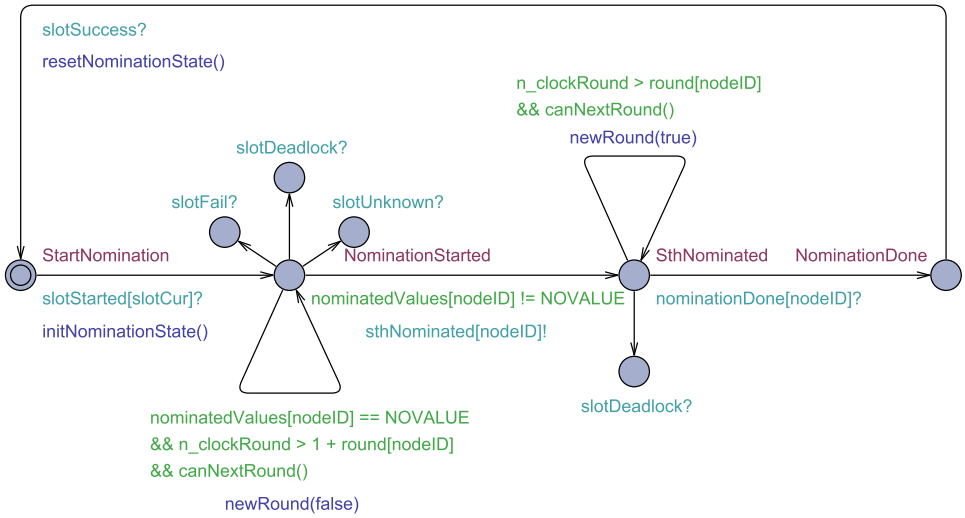


Fig. 4. The Template for the Nomination Protocol

state of the node cannot go backward. When a node handles its receiving messages, it retrieves the stored messages for every node in its validators without the concern of duplicated or disordered messages. Then, the message handlers for each node are executed in a randomized manner, and this allows for modeling the similar behaviors of the asynchronous network with significantly reduced state space.

B. Nomination Protocol

The template for the nomination protocol illustrated in Fig. 4 consists of two parts: the 'NominationStarted' state (performing the federate voting with input values for a nominated value) and the 'SthNominated' state (exchanging the messages for already introduced extra nominated values without introducing new input values). The nomination template starts after a slot is started by the slot manager though the 'slotStarted[slotCur]' channel. In each round, every intact node (a node that behaves as expected by the protocol) selects a node with the highest priority in that round. The selected node is supposed to be identical for all intact nodes. The node with the highest priority can introduce a new value for the federate voting. Then, the other nodes will vote for the values introduced by the highest priority node. Once a node recognizes a value that is agreed for the nomination, the node initiates its ballot template through the 'sthNominated[nodeID]' channel, which acts like a synchronization point between the nomination template and the ballot template. Then, the node transits to the 'SthNominated' state, where the nodes only exchange the messages for the agreement of known values. The transition of the ballot protocol from the 'Prepare' state to the 'Commit' state triggers the state transition of the nomination template to the 'NominationDone' state.

There are three possible failures for the nomination template: 'slotFail', 'slotUnknown' and 'slotDeadlock'. The 'slotFail' channel is triggered when the quorums in the system are not healthy (if the quorum intersections are broken or the number of intact nodes in a quorum is less than its threshold). The other two failures are related to checking the issues created by the parameterization (details are provided in Section IV).

C. Ballot Protocol

The template for the ballot protocol in Fig. 5 has three states: 'Prepare,' 'Commit,' and 'Externalize.' During either the 'Prepare' state or the 'Commit' state, the ballot template periodically checks messages, which contain the states of other nodes, to agree on the nominated values. The specification of the implementation [9] mentioned that every node in the 'Externalize' state must broadcast its state for a certain amount of time to help others agree on the externalized value. However, our network model does not require the broadcasting messages because other nodes can refer to the status of the externalized nodes from the global message store. The broadcasting messages are required if the protocol is modeled using a different network model.

The execution of the ballot template continues until the slot manager template reaches either the deadlock state or a successful agreement between nodes. If all intact nodes in the 'Externalize' state have the identical value in their ballots, then the current slot is considered a success and the template returns to the initial state for the next slot. The presence of different values in the 'Externalize' states of the nodes in a quorum implies failure in reaching an agreement and the failure of the current slot.

D. Slot Manager

Each slot in our model represents a block in a blockchain because the agreed value in a slot can be combined with multiple input values (a list of transactions for the cryptocurrency). A new slot is started when the previous slot is successfully agreed, i.e., all intact nodes in a quorum externalize the identical value in the previous slot and the number of the intact nodes is larger than the quorum's threshold.

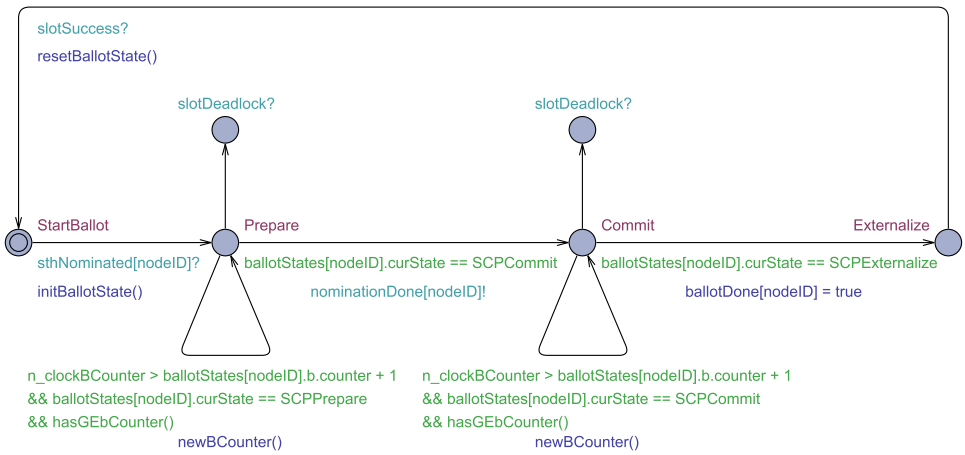


Fig. 5. The Template for the Ballot Protocol

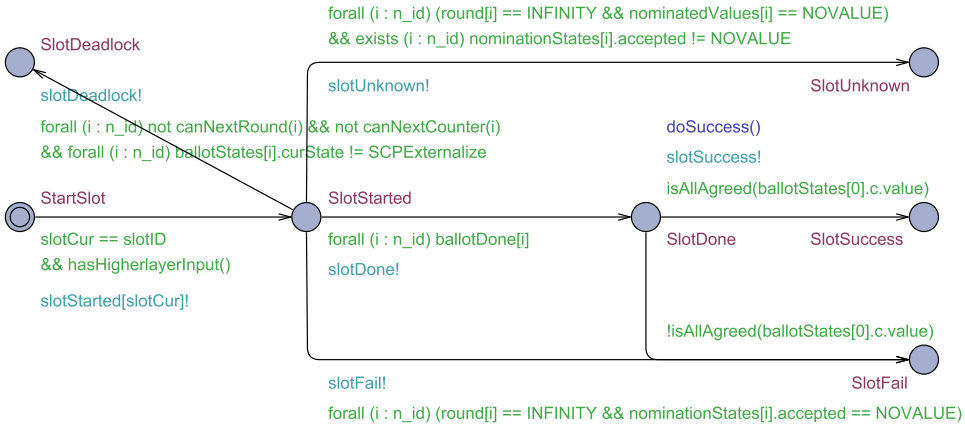


Fig. 6. The Template for the Slot Manager

The slot manager template shown in Fig. 6 has two responsibilities: a) managing multiple slots for the SCP and b) observing the failures caused by invalid parameters. The slot manager template can be terminated with four possible outcomes: 'slotSuccess,' 'slotFail,' 'slotUnknown,' and 'slotDeadlock.' The trigger of the 'slotSuccess' channel implies successful agreement between more-than-threshold intact nodes, and this moves the execution to the next slot. The nomination and ballot templates reset all internal data before the next slot starts. The 'slotFail' channel is triggered when a) the nomination protocol cannot agree on any votes until it reaches the maximum round (details are provided in Section IV) or b) the externalized values are different at certain nodes after completing the ballot template. The 'slotUnknown' and 'slotDeadlock' channels are related to the parameterization described in Section IV, and the detailed explanation is provided in Section IV.

IV. Formal Verification

In this section, we will explain the types of properties that are guaranteed in our model for the verification. Additionally, we will show how the state space and the execution time of the model is reduced by the parameterization. Finally, a detailed discussion regarding the results of the verification will be presented.

A. Parameterization for the State Space Reduction

The differences between the clock in timed automata and that in real systems could result in unintended behaviors to our model. This could be one of the major causes for the state explosion problem during the verification. Both sub-protocols in SCP (the nomination and ballot protocols) use timers to increase the chances of staying in the same context for nodes in the system: the same round number for the nomination protocol and the same ballot counter for the ballot protocol. It is possible that a node continues to increase its round number in the nomination protocol while others do nothing, because each clock for individual nodes works independently in timed automata (as is the case of our model). Additionally, these round numbers or ballot counters for the nodes could increase indefinitely until it attains the maximum value of the integer, 32767. To avoid the indefinitely-increasing timer issue, we added the first parameter, INFINITY, into our model.

Fig. 7 shows code snippets in our model, including the definition and usage of the first parameter related to the limitation

```
// the maximum limit for round numbers
//     and ballot counters
const int INFINITY = 5;

// declarations for timers
int[0,INFINITY] round[N];
int[0,INFINITY] counter;

...

// for nomination protocol
if (round[nodeID] < INFINITY) {
    round[nodeID]++;
}

...

// for ballot protocol
if (ballotStates[nodeID].b.counter
        < INFINITY) {
    ballotStates[nodeID].b.counter++;
}
```

Fig. 7. Code Snippets for the Parameterization

of the timers in the nomination and the ballot protocols. Both timers can be increased until they reach the limit. Furthermore, both protocols maintain the maximum value for each timer. This allows for the declaration of the variables of the timers with their range. The ranged integer values can reduce the state space significantly, compared to integers without a range (from -32768 to 32767). However, this limitation of the timer might alter the behaviors of the protocols because it is usually not expected that the nodes remain in the maximum timer value forever. Our experiment shows that having a small value for INFINITY results in agreement being achieved too early as the longer that nodes stay in the same round or in the same ballot counter, the sooner the agreement is completed. The 'slotUnknown' channel is for the detection of this false agreement. The 'slotUnknown' channel will be triggered when the agreement is completed, which occurs after all the nodes have already reached the INFINITY value for their timers.

The timer limit INFINITY value can reduce the state space by setting the ranges of the timers; however this limit cannot prevent the behavior that causes the state explosion – a single node keeps firing its timer and other nodes are idle. This situation can occur because time passes differently in timed automata. The values for the timers increase by one after the timers are fired, which can result in the probability of keeping the same round number and the same ballot counter becoming high. Therefore, the unevenly-progressed timers among nodes could significantly delay the process of reaching the agreement. Furthermore, this will also increase the usage of the state space for the verification.

The second parameter, MAXGAP, prevents a single node from advancing too far without considering other nodes in its quorum. A node is only allowed to check received messages (in our model, it checks the global message store) when the difference between the timer value of the node and the minimum timer value of other nodes in the quorum is smaller than the value of MAXGAP. Using a small value for the MAXGAP parameter could significantly reduce the state space for the verification; however, MAXGAP, similar to INFINITY, can also cause a new problem if the value of MAXGAP is too small - deadlock (no node is allowed to check received messages). To determine whether the deadlock is caused by a small MAXGAP value, the 'slotDeadlock' channel monitors the cases where all intact nodes are not allowed to check received messages.

B. Properties for Verification

Fig. 8 shows a list of queries to check the properties of our model. The first four properties are related to the correctness of SCP (safety and liveness). The last two properties are designed to determine the correct parameters in the model. Query (1) is used to find an execution to reach a successful agreement for all slots and shows the execution for the simulation purposes.

The nomination protocol in SCP should maintain the following invariant: *once a node has a value for the nominated candidates, the entire intact nodes in its quorum should have the same value for the nominated candidates*. The template in Fig. 9 is an observer template to check whether the Nomination template maintains this invariant. The invariant is encoded in Query (2). This means that the Nomination Observer template cannot reach the 'CheckNominationFail' state. Query (3) is the following invariant that the ballot protocol should maintain: *if 'c' ballot is not a NULL ballot at any nodes, then it ensures 'c' ballot* $\lesssim$ *'h' ballot* $\lesssim$ *'b' ballot*.

Query (4) verifies that all executions do not reach the 'SlotFail' state in the Slot Manager template. The 'SlotFail' state can be reached when a) the Nomination template cannot agree on a single federate vote until the round numbers of the entire intact nodes in a quorum reach the INFINITY value or b) the Ballot template is ended with different external values.

If the first parameter, INFINITY, has a smaller value than that required to reach an agreement, then Query (5) will not be satisfied. This query is useful to identify the smallest value for INFINITY, that can achieve the completion of the verification with the reduced state space. The second parameter, MAXGAP, usually has the value '1', which is the tightest gap value. Query (6) checks whether a given quorum configuration is safe so that there is no deadlock.

C. Verification for Configurations with Full Quorums

Table I presents the results of the verification for the quorum configurations with Full Quorums. The word 'Full Quorums' means that all quorums in the full quorums are connected with quorum intersections and all nodes in each quorum are well-connected (all nodes have all members of the quorum in their quorum configurations). The verification was performed using UPPAAL version 4.1.19 on Ubuntu 18.04.1 LTS machine with an Intel Core i7 3.2 GHz CPU and 64 GB RAM. The first column has the indices for each quorum configuration and the verification results of the configurations are denoted by these indices. The second column contains a

$$E <> \ forall\ (i : int[0, SLOTMAX - 1])\ NodeSlot(i).SlotSuccess \quad (1)$$

$$A[\,]\ forall\ (i : vId)\ not\ NodeCheckNomination(i).CheckNominationFail \quad (2)$$

$$\begin{aligned} A[\,]\ & forall\ (i : nId)\ ballotStates[i].c.counter! = 0\ imply \\ & (ballotStates[i].c.value == ballotStates[i].h.value\ \&\& \\ & \quad ballotStates[i].c.counter <= ballotStates[i].h.counter)\ \&\& \\ & (ballotStates[i].h.value == ballotStates[i].b.value\ \&\& \\ & \quad ballotStates[i].h.counter <= ballotStates[i].b.counter) \end{aligned} \quad (3)$$

$$A[\,]\ forall\ (i : sId)\ not\ NodeSlot(i).SlotFail \quad (4)$$

$$A[\,]\ forall\ (i : sId)\ not\ (NodeSlot(i).SlotFail\ ||\ NodeSlot(i).SlotUnknown) \quad (5)$$

$$A[\,]\ forall\ (i : sId)\ not\ (NodeSlot(i).SlotFail\ ||\ NodeSlot(i).SlotUnknown\ ||\ NodeSlot(i).SlotDeadlock) \quad (6)$$

Fig. 8. Verification Queries.

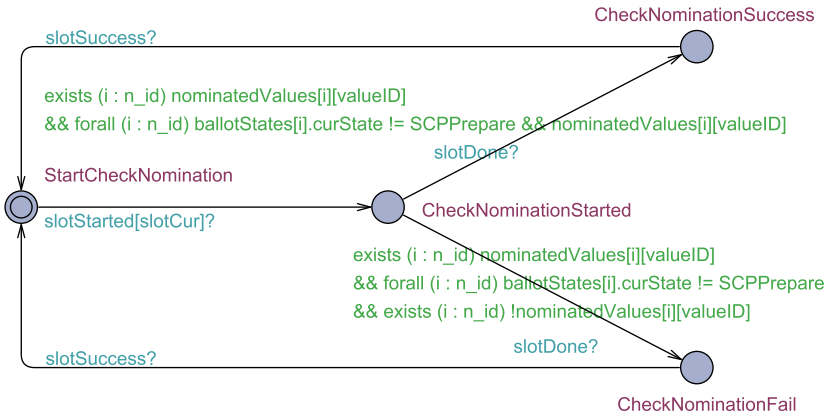


Fig. 9. Template for the Observer of the Nomination Protocol

list of quorum configurations. The third column shows the values for INFINITY and the forth column shows the values for MAXGAP. The six columns for the Queries (from the fifth to the tenth columns) show how much the state space was used and for how long it was executed for the verification.

It is noticeable that configurations 2 and 3 have values which are larger than those in configuration 1 for both parameters (one for INFINITY and the other for MAXGAP) with an identical quorum configuration as configuration 1. The comparison of the state space and the execution time among configurations shows the importance of finding small values for the parameters, especially for MAXGAP. Increasing the values for MAXGAP also requires the increment of the ones for INFINITY to avoid a deadlock. Owing to the increase in both parameters, configuration 3 uses nearly 300 times more state space than the one for configuration 1. Table I lists that the number of nodes in a quorum. The number of quorums in the quorum configuration are strongly related to the state space and the execution time of the verification. For example, configuration 7 has two more nodes than configuration 1 and uses 2300 times more state space than the one of configuration 1. Additionally, the execution time for the configuration with 5 nodes (configuration 7) took more than 5 hours and that with 3 nodes (configuration 1) took 0.49 s.

D. Verification for Configurations with Partial Quorums

Determining whether a given quorum configuration has quorum intersections and satisfies the theorem of SCP [1] is difficult, because each node only has information regarding its quorum slice as the quorum configuration instead of the entire quorum. Additionally, it is easy to build a quorum with partially-connected nodes (a partial quorum) such as the quorum configuration in Fig. 10. Nodes 1 and 2 have all nodes (node 0, 1, 2, 3) in their quorum configurations; however, nodes 0 and 3 have a missing node in their configuration. The quorum configuration in Fig. 11 shows an example of a Full Quorum. Each node has all nodes in its quorum configuration.

Owing to the difference in the connectivity between both quorum configurations, the two quorum configurations result in different outcomes for Query (6) even though they construct

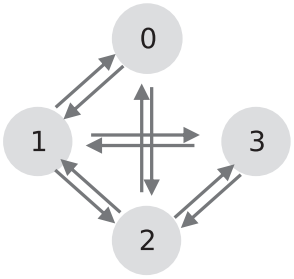
TABLE I
VERIFICATION RESULTS FOR THE CONFIGURATIONS WITH FULL QUORUMS

#	Configuration	INF	MAX	**Query** (1)	**Query** (2)	**Query** (3)	**Query** (4)	**Query** (5)	**Query** (6)
1	3 Nodes 1 Quorum	5	1	13.45 MB 0.41 s	13.91 MB 0.49 s	13.95 MB 0.53 s	13.90 MB 0.52 s	13.91 MB 0.53 s	13.92 MB 0.56 s
2	3 Nodes 1 Quorum	9	2	76.16 MB 18.09 s	369.97 MB 105.88 s	370.01 MB 107.3 s	321.34 MB 105.42 s	369.97 MB 106.03 s	369.98 MB 106.31 s
3	3 Nodes 1 Quorum	13	3	213.29 MB 55.63 s	4.02 GB 1,301.64 s	4.02 GB 1,338.21 s	4.02 GB 1,303.22 s	4.02 GB 1,288.89 s	4.02 GB 1,302.34 s
4	4 Nodes 1 Quorum	5	1	136.30 MB 46.67 s	201.64 MB 72.74 s	201.66 MB 73.4 s	201.64 MB 74 s	201.64 MB 73.54 s	201.64 MB 72.92 s
5	4 Nodes 2 Quorums	5	1	784.86 MB 223.52 s	1.09 GB 350.97 s	1.09 GB 355.54 s	1.09 GB 351.43 s	1.09 GB 354.38 s	1.09 GB 352.81 s
6	5 Nodes 1 Quorum	6	1	2.48 GB 1,816.99 s	4.45 GB 3,617.96 s	4.45 GB 3,589.64 s	4.45 GB 3,588.55 s	4.45 GB 3,586.8 s	4.45 GB 3,584.01 s
7	5 Nodes 2 Quorums	6	1	15.97 GB 8,257.14 s	32.44 GB 18,764.21 s	32.44 GB 19,014.01 s	32.44 GB 18,776.65 s	32.44 GB 19,212.85 s	32.44 GB 18,671.09 s
8	4 Nodes 2 Quorums	5	1	407.79 MB 110 s	NOT SATISFIED	412.98 MB 114.87 s	NOT SATISFIED	NOT SATISFIED	NOT SATISFIED

```
// total number of nodes
const int N = 4;

SCPQuorumSlice qs[N] =
    {{0, {0, 1, 2, N}},
     {0, {0, 1, 2, 3}},
     {0, {0, 1, 2, 3}},
     {0, {N, 1, 2, 3}}};
```

(a) Textual representation



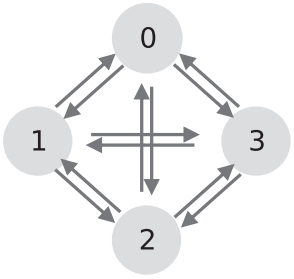
(b) Figure

Fig. 10. A Partial Quorum with 4 nodes

```
const int N = 4;

SCPQuorumSlice qs[N] =
    {{0, {0, 1, 2, 3}},
     {0, {0, 1, 2, 3}},
     {0, {0, 1, 2, 3}},
     {0, {0, 1, 2, 3}}};
```

(a) Textual representation



(b) Figure

Fig. 11. A Full Quorum with 4 Nodes

the same quorum. By the definition of a quorum in [1], Fig. 10 and Fig. 11 show the same quorum with four nodes: node 0, 1, 2, and 3. The successful verification results of the quorum configuration in Fig. 11 are shown in the row for configuration 4 in Table I. However, the quorum configuration in Fig. 10 could not satisfy Query (6) no matter how large the INFINITY value is even though MAXGAP has the smallest value, '1'.

E. Verification for Configurations without Quorum Intersections

The verification results for configuration 8 in Table I shows that a quorum configuration without quorum intersections cannot reach an agreement even though all quorums in the quorum configuration are Full Quorums. The detailed information about configuration 8 is presented in Fig. 12 (two Full Quorums without quorum intersections). Configuration 8 could have a successful agreement (from the result of Query (1)) but it cannot reach a successful agreement always (Query (4) is not satisfied). Those results satisfy the theorem in [1] - the successful agreement can only be guaranteed with quorum intersections. In addition, the result of Query (2) shows that the invariant for the Nomination protocol cannot be satisfied without quorum intersections.

V. Discussion

A. Timing

The parameterization in our model is introduced to resolve the unexpected behaviors caused by the difference between the clocks of the timed automata and actual systems. UPPAAL resolves timing constraints with clocks. Every clock in UPPAAL has the same speed. That is, if a clock increases by a certain amount, then other clocks also increase by the same amount. The protocols in the SCP increase the probability of reaching an agreement by maintaining the same node number and ballot counter. However, the operation of the clock in UPPAAL cannot prevent a single node from advancing alone. The MAXGAP parameter can manage the progress of each node and this could increase the chance for nodes to stay in the similar clock.

B. Scalability

The state explosion problem is a major challenge in the application of the model checking technique. Sound abstraction with respect to each property is required to verify a model with large state space. In our study, one major source of state space explosion is asynchronous communication among nodes. Restricting fully asynchronous communication among nodes by providing MAXGAP and INFINITY as parameters to each model is a method of reducing state space. However, the number of nodes that can be dealt with by the UPPAAL model checker is not very large. UPPAAL provides the advantage of resolving timing constraints, but it is not considered as a highly efficient model checker. Even though we could not examine other model checkers in this study, we have a plan to utilize other model checkers to solve the problem of consensus protocol modeling and verification.

C. Applicability and Extensibility

This work presents a UPPAAL timed automata model for a specific protocol, i.e., the SCP. Other Byzantine agreement-based consensus protocols, such as Algorand [10], can possibly obtain insights from our work in their modeling and verification.

The presented UPPAAL model is considered to be considerably larger than other UPPAAL models in relevant literature. The abstraction techniques we used for our modeling and verification could provide useful insights to other UPPAAL users.

The network model of this study are considered to present the similar behaviors of the asynchronous network. However, various issues in exchanging messages in the asynchronous network, such as failing to deliver messages and delaying to deliver messages, were not fully considered in the network model. In order to verify the achievement of the successful consensus considering the real network issues, a different network component can be modeled in separate templates in UPPAAL and can be included in the system declaration.

VI. Related Work

To the best of our knowledge, there is no study on modeling and verification of the SCP to investigate and evaluate quorum configurations. However, several researchers have studied related topics.

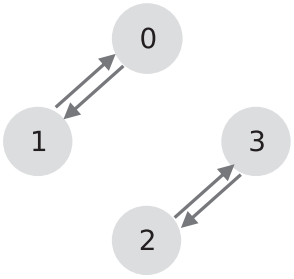
Duan et al. [11] presented the model-based formalization, simulation, and verification of a blockchain system, with a focus on a certain consensus algorithm (i.e., an improved Byzantine fault tolerant algorithm [12]), using Specification Description Language (SDL). While they showed that model-based verification using SDL works for a certain consensus algorithm, they did not consider timing constraints, which can be critical in realistic consensus.

A large number of researchers summarized the consensus algorithms used in blockchains. Bach et al. [13] compared several consensus algorithms in terms of algorithmic analysis, including Proof of Work (PoW), Proof of Stake (PoS), Ripple protocol consensus algorithm (RPCA), SCP, delegated Proof of Stake (dPoS), and Proof of Importance (PoI), in terms of algorithmic analysis. They compared the energy saving, tolerated power of adversary, and scalability of the consensus algorithms. Cachin and Vukolič [14] summarized blockchain consensus protocols, including Hyperledger Fabric, Tendermint, Symbiont, R3 Corda, Iroha, Kadena, Chain, Quorum, MultiChain, Sawtooth Lake, Ripple, Stellar, and IOTA, focusing on permissioned systems in the sense that their participants are identified. Sankar et al. [15] performed similar comparative analysis on SCP, Corda, and Hyperledger. Dinh et al. [16] provided an in-depth survey of blockchain systems and a benchmark framework for performance investigation against data processing workloads. Innerbichler and Damjanovic-Behrendt [17] recently explored the use of the SCP and its FBA algorithm for ensuring trust and reputation between federated, cloud-based platform instances (nodes) and

```
const int N = 4;

SCPQuorumSlice qs[N] =
    {{0, {0, 1, N, N}},
     {0, {0, 1, N, N}},
     {0, {N, N, 2, 3}},
     {0, {N, N, 2, 3}}};
```

(a) Textual representation



(b) Figure

Fig. 12. Two Quorums without Quorum Intersections

their participants. They identified major components for the implementation of the SCP in federated ecosystems.

Several studies provided formal verification on smart contracts, which is another fundamental basis of a blockchain system.

Decentralised smart contracts represent the next step in the development of protocols that support the interaction of independent players without the presence of a coercing authority. Based on protocols la BitCoin for digital currencies, smart contracts are believed to be a potentially enabling technology for a wealth of future applications. The validation of such an early developing technology is as necessary as it is complex. In this paper we combine game theory and formal models to tackle the new challenges posed by the validation of such systems.

Bhargavan et al. [18] studied smart contracts in a blockchain system using formal verification. Their preliminary experiments using $\mathtt{F}^\star$ to verify smart contracts suggest that the type and effect system of $\mathtt{F}^\star$ is sufficiently flexible for capturing and proving the properties of interest for contract programmers. Bai et al. [19] recently proposed a general smart contract template and verified deadlock and livelock in a small shopping smart contract model using SPIN [20]. Bigi et al. [21] utilized formal verification with game theory to analyze and validate an idealized protocol for decentralized smart contracts. Wang et al. [22] used PRISM [23] to verify the effectiveness of the introduction of randomness in Criminal Smart Contracts (CSCs). They found that such randomness factors can decrease the power of CSCs. Annenkov and Elsman [24] introduced verifiable programming languages for smart contracts and used the Coq proof asistant to prove the desirable properties, such as soundness, of smart contracts.

However, none of these studies performed formal verification of consensus algorithms in a blockchain system.

VII. Conclusion

We developed a UPPAAL timed automata model for the SCP, which is an FBA-based consensus protocol for blockchain platforms. We simulated the model under various quorum configurations and verified it using a model checking technique. Through the modeling and verification of the SCP, we could check whether a certain quorum configuration ensures consensus or not, before execution on an actual network. We presented abstraction techniques that help in coping with the extremely large state space of the SCP and the environment model in formal verification. The proposed modeling and verification techniques and practices can possibly be utilized for other consensus protocols of various blockchain platforms.

As UPPAAL has limited scalability for asynchronous protocol verification, we plan to model and verify the SCP using other verification tools and compare the results. We also plan to analyze various blockchain consensus algorithms through modeling and verification.

Acknowledgment

This research was supported by BlockchainOS Inc. and Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2016R1A6A3A11931209).

References

[1] D. Mazieres, "The stellar consensus protocol: A federated model for internet-level consensus," *Stellar Development Foundation*, 2015.

[2] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on UPPAAL," in *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, ser. LNCS, M. Bernardo and F. Corradini, Eds., no. 3185. Springer–Verlag, September 2004, pp. 200–236.

[3] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *International journal on software tools for technology transfer*, vol. 1, no. 1-2, pp. 134–152, 1997.

[4] J. Bengtsson, W. D. Griffioen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Verification of an audio protocol with bus collision using uppaal," in *International Conference on Computer Aided Verification*. Springer, 1996, pp. 244–256.

[5] K. Havelund, A. Skou, K. G. Larsen, and K. Lund, "Formal modeling and analysis of an audio/video protocol: An industrial case study using uppaal," in *Real-Time Systems Symposium, 1997. Proceedings., The 18th IEEE*. IEEE, 1997, pp. 2–13.

[6] A. P. Ravn, J. Srba, and S. Vighio, "A formal analysis of the web services atomic transaction protocol with uppaal," in *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. Springer, 2010, pp. 579–593.

[7] ——, "Modelling and verification of web services business activity protocol," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2011, pp. 357–371.

[8] J. H. Kim, K. G. Larsen, B. Nielsen, M. Mikučionis, and P. Olsen, "Formal analysis and testing of real-time automotive systems using uppaal tools," in *International Workshop on Formal Methods for Industrial Critical Systems*. Springer, 2015, pp. 47–61.

[9] N. Barry, G. Losa, D. Mazieres, J. McCaleb, and S. Polu, "The Stellar Consensus Protocol (SCP)," Internet Engineering Task Force, Internet-Draft draft-mazieres-dinrg-scp-05, Nov. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-mazieres-dinrg-scp-05

[10] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: ACM, 2017, pp. 51–68. [Online]. Available: http://doi.acm.org/10.1145/3132747.3132757

[11] Z. Duan, H. Mao, Z. Chen, X. Bai, K. Hu, and J.-P. Talpin, "Formal modeling and verification of blockchain system," in *Proceedings of the 10th International Conference on Computer Modeling and Simulation*, ser. ICCMS 2018. New York, NY, USA: ACM, 2018, pp. 231–235. [Online]. Available: http://doi.acm.org/10.1145/3177457.3177485

[12] Z. Chen, "Research on private blockchain based on crowdfunding [j]," *Journal of Information Security Research*, vol. 3, no. 3, pp. 227–236, 2017.

[13] L. Bach, B. Mihaljevic, and M. Zagar, "Comparative analysis of blockchain consensus algorithms," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 2018, pp. 1545–1550.

[14] C. Cachin and M. Vukolic, "Blockchain consensus protocols in the wild," *CoRR*, vol. abs/1707.01873, 2017. [Online]. Available: http://arxiv.org/abs/1707.01873

[15] L. S. Sankar, M. Sindhu, and M. Sethumadhavan, "Survey of consensus protocols on blockchain applications," in *Advanced Computing and Communication Systems (ICACCS), 2017 4th International Conference on*. IEEE, 2017, pp. 1–5.

[16] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling blockchain: A data processing view of blockchain systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 7, pp. 1366–1385, July 2018.

[17] J. Innerbichler and V. Damjanovic-Behrendt, "Federated byzantine agreement to ensure trustworthiness of digital manufacturing platforms," in *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems*, ser. CryBlock'18. New York, NY, USA: ACM, 2018, pp. 111–116. [Online]. Available: http://doi.acm.org/10.1145/3211933.3211953

[18] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Gollamudi, G. Gonthier, N. Kobeissi, N. Kulatova, A. Rastogi, T. Sibut-Pinote, N. Swamy, and S. Zanella-Béguelin, "Formal verification of smart contracts: Short paper," in *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, ser. PLAS '16. New York, NY, USA: ACM, 2016, pp. 91–96. [Online]. Available: http://doi.acm.org/10.1145/2993600.2993611

[19] X. Bai, Z. Cheng, Z. Duan, and K. Hu, "Formal modeling and verification of smart contracts," in *Proceedings of the 2018 7th International Conference on Software and Computer Applications*, ser. ICSCA 2018. New York, NY, USA: ACM, 2018, pp. 322–326. [Online]. Available: http://doi.acm.org/10.1145/3185089.3185138

[20] G. J. Holzmann, "The model checker spin," *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295, May 1997.

[21] G. Bigi, A. Bracciali, G. Meacci, and E. Tuosto, *Validation of Decentralised Smart Contracts Through Game Theory and Formal Methods*. Cham: Springer International Publishing, 2015, pp. 142–161.

[22] Y. Wang, A. Bracciali, T. Li, F. Li, X. Cui, and M. Zhao, "Randomness invalidates criminal smart contracts," *Information Sciences*, vol. 477, pp. 291 – 301, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025518308740

[23] M. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: Verification of probabilistic real-time systems," in *Computer Aided Verification*, G. Gopalakrishnan and S. Qadeer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 585–591.

[24] D. Annenkov and M. Elsman, "Certified compilation of financial contracts," in *Proceedings of the 20th International Symposium on Principles and Practice of Declarative Programming*, ser. PPDP '18. New York, NY, USA: ACM, 2018, pp. 5:1–5:13. [Online]. Available: http://doi.acm.org/10.1145/3236950.3236955