Byzantine Fault Tolerance in Distributed Systems: Advancing the Replica State Discovery Protocol v2.0*

Maksym Kotov[1,†], Serhii Toliupa[1,†], Volodymyr Nakonechnyi[1,†], Serhii Buchyk[1,†] and Oleksandr Buchyk[1,*,†]

[1] *Taras Shevchenko National University of Kyiv, 60 Volodymyrska str., 01033 Kyiv, Ukraine*

Abstract

Contemporary demands for computational complexity and fault tolerance have led to the development of different approaches toward building replicated and clustered environments. The foundational aspect of these systems is a set of underlying consensus, management mechanisms, and protocols. It is quite common to have a centralized management plane responsible for deployment, task assignment, and result aggregation to avoid complexities intrinsic within a distributed consensus paradigm. However, the fault tolerance and trust decentralization capabilities of such an approach remain restricted. One of the most prominent examples of a distributed system is blockchain technology and its off-chain networks introducing capabilities for managing and coordinating the assignment of computational tasks. Blockchain has its limitations, mainly related to response times and throughput, as it necessitates consensus for every action and interaction. A lightweight cluster coordination and consensus management framework, the Replica State Discovery Protocol (RSDP), aims to provide rapid coordination of nodes. RSDP defines an interface for arbitrary logical extension of distributed computation modules and establishes a set of rules for nodes to follow to achieve consensus within the network. Nonetheless, RSDP was initially designed as a protocol for private coordination, and in its conception, it was not constructed with Byzantine fault tolerance (BFT) in mind. The purpose of this paper is to advance the said coordination method to incorporate practices that allow for secure decentralized computation coordination even in the presence of malicious actors. Firstly, this article defines methods to achieve strict state transitions that avoid trust exploitations and flooding techniques. Secondly, this paper presents multiple approaches toward building strict quorum state reducers within RSDP, which allow to initiation of BFT-compliant operations. Thirdly, an additional set of new mechanisms and methods are described in the context of RSDP to improve both efficiency and reliability. Finally, we propose the generalized BFT model for RSDP, which leverages blockchain as a trusted source and enables the establishment of a completely decentralized public coordinated computing environment.

Keywords

computer network, network protocol, coordination of distributed system, distributed consensus, Byzantine fault tolerance, BFT, Replica State Discovery Protocol, RSDP, BFT-compliant consensus within RSDP

1. Introduction

The rising popularity of Internet technology has been rapidly increasing over the past few decades. Nowadays, online interactions between remotely located parties have become quite common in both business and governance sectors. Distributed systems serve as a technical foundation for such operations [1, 2]. The demand led to numerous research and engineering efforts to improve the security, reliability, and availability characteristics of these systems [3].

One of the primary aspects that define architectural complexity is the coordination mechanism. Depending on the nature of managed nodes, the task of cluster management tends to be nontrivial and requires custom solutions to achieve state synchronization, failover, and availability. In that context, there are two approaches based on the type of managed services: replication and

*CPITS 2025: Workshop on Cybersecurity Providing in Information and Telecommunication Systems, February 28, 2025, Kyiv, Ukraine

* Corresponding author.

† These authors contributed equally.

maksym_kotov@ukr.net (M. Kotov); tolupa@i.ua (S. Toliupa); nvc2006@i.ua (V. Nakonechnyi); buchyk@knu.ua (S. Buchyk); alex8sbu@knu.ua (O. Buchyk)

0000-0003-1153-3198 (M. Kotov); 0000-0002-1919-9174 (S. Toliupa); 0000-0002-0247-5400 (V. Nakonechnyi); 0000-0003-0892-3494 (S. Buchyk); 0000-0001-7102-2176 (O. Buchyk)

clusterization, which respectively correspond to stateless and stateful endpoints united under a single abstract amalgamation [4–11].

The replication approach applies to networks responsible for managing a set of homogeneous multiagent systems [4–6]. Such installments are characterized by the interchangeability of the constituents; that is, every participating node can be substituted by another without disrupting the overall performance or operational stability of a distributed system. In that case, internal architecture is commonly composed of an external-facing request forwarder and an internal pool of replaceable nodes that perform the desired computation. A monitoring solution continuously probes the active nodes, and upon detection of any inconsistency or response delays, signals the control plane to simply redeploy the faulty instance.

On the other side, clusterization refers to a method of organizing, managing, and maintaining coordination between a set of heterogeneous multiagent systems. Clustered environments are often comprised of either a set of stateful nodes or nodes responsible for handling unexchangeable procedures [9–11]. The said approach requires significant engineering effort to effectively manage request distribution and consistent operation within the system. In that case, it is quite common to represent the interactions in two abstract layers: coordination and execution.

The coordination layer taxonomy includes decentralized and centralized systems, providing publicly and privately governed mechanisms respectively. Centralized systems are also characterized by a single coordination responsibility service, where a server or a group of servers oversee and reconcile operations within a network under regulated control [12, 13]. In turn, decentralized systems represent a network of participating peers coordinated through a consensus mechanism. We could further divide these into classes based on their adherence to the Byzantine fault tolerance principles, which outline the inherent complexities associated with open networks.

Having said that, the Replica State Discovery Protocol (RSDP) is the first consensus framework developed for coordination of the clustered and replicated decentralized systems [14, 15]. It defines the abstraction layers and foundations, upon which a plethora of arbitrary computational logic could be implemented to achieve lightweight, consistent, and coordinated execution management without requirements for a central responsible entity.

One of the prominent examples of contemporary decentralized technologies compliant with BFT is the blockchain [16–19]. Blockchain has recently become widespread, growing in interest rates and demand from its substantial user base worldwide. This technology defines a logical basis for verifiable public asset transfer and computation crucial to establishing a secure transactional environment. However, it is quite limited in its scalability and overall throughput, which is an area of active research [20–23].

Initially, RSDP was designed as a decentralized coordination solution to be used within private permissioned networks. The protocol assumes that a secure permissioned environment is provisioned for the cluster. Its primary role is to rapidly reconcile cluster-wide state and, as an implication, dynamically provide all the necessary context information for cluster members to perform their procedures. RSDP was not designed to withstand any potential state transition violations or malicious interactions and thus is not compliant with the BFT requirements.

The purpose of this article is to advance RSDP capabilities to the 2.0 version, which introduces a secure coordination layer required for both unstable and malicious environments. To achieve that, firstly, this article introduces a set of new phase transition controls to restrict possible abuse of participating nodes. In addition, this paper outlines a new consensus quorum approach defined on the state reducers level. The mechanisms allow to improve protocols' resistance capabilities against both network losses and potential outlying coordination impact. Lastly, this article introduces a novel approach toward open decentralized computation coordination based on RSDP and blockchain technologies [24].

2. Overview of the replica state discovery protocol

In its basis, RSDP relies on the communication layer responsible for handling message-sending and receiving procedures. Originally, the protocol defined an abstraction layer built on top of AMQP, called a Simulated Local Area Network (SLAN). This abstraction allows to simplify the development of cluster-wide network protocols and is described in its dedicated article [14].

The SLAN logical topology is shown in Fig. 1:

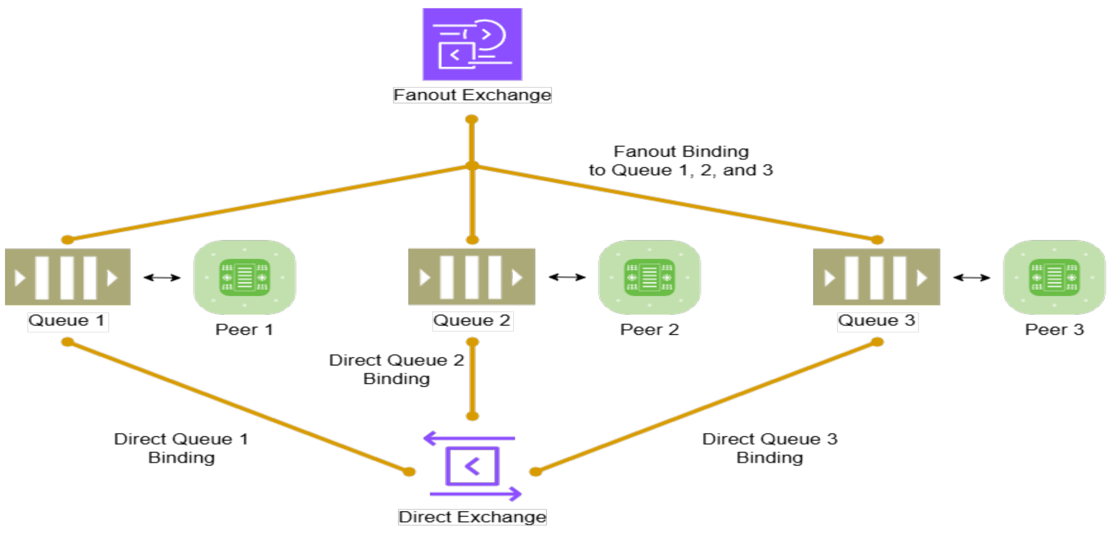

Figure 1: SLAN network topology

The SLAN network topology is comprised of participating nodes, their respective queues, and a central coordination entity responsible for initiating and managing communication channels. That is, a central coordination is an AMQP server, where one of the most popular and widely used implementations is RabbitMQ [25–27].

In its paradigm, the communication process is designed to be reliable, scalable, and asynchronous. The message producer does not address the recipient but rather the queue or a set of queues when initiating a message-passing procedure. In the case of SLAN, this capability is leveraged to both establish a broadcast basis and a direct message sending by assigning anonymous dynamic queues to each peer individually.

To begin with, let us define the core entities of SLAN:

- ***Nodes:*** $N=\{n_1,\ldots,n_{|N|}\}$, where each $n\in N$ is a network participant that can send and receive messages.
- ***Messages:*** $M=\{m_1,\ldots,m_{|M|}\}$, each $m\in M$ is a data payload transmitted among nodes.
- ***Queues:*** $Q=\{q_1,\ldots,q_{|Q|}\}$ with a bijection $Q:N\rightarrow Q$. Each node n has a unique queue $Q(n)$ holding messages before consumption.

In the context of SLAN, the individual queues are designed to be transient and accessible only within the confines of the established connection between the peer and the AMQP server. That approach allows for simplified dynamic handling of joining nodes but also lacks durability guarantees for messages.

The local area networks provide two basic communication operations: broadcast and direct message passing. Having these operations allows us to build advanced custom protocols. However, it is quite a challenge to provide these primitives within distributed systems that span multiple networks or even regions. By leveraging AMQP capabilities, SLAN allows for establishing such a basis efficiently and securely by leveraging direct and fanout exchange types defined in the underlying protocol.

We can define the routing structures as follows:

- ***Exchanges:*** $E = \{E_{\text{fanout}}, E_{\text{direct}}\}$, where specific exchanges route messages: E_{fanout} broadcasts to all queues, E_{direct} routes by a key.
- ***Fanout Bindings:*** $\forall q \in Q, (E_{\text{fanout}}, q) \in B_{\text{fanout}}$, where every queue is bound to E_{fanout}.
- ***Routing Keys:*** K is the set of keys, and $R: N \to K$ is injective, where each node n is assigned a unique key $R(n)$ for direct routing.
- ***Direct Bindings:*** $(E_{\text{direct}}, Q(n), R(n)) \in B_{\text{direct}}$ for each $n \in N$, where the direct exchange maps each key $R(n)$ to the corresponding queue $Q(n)$.

Within AMQP, routing keys play a pivotal role in its ability to establish complex interaction schemas between exchanges and queues. For instance, the protocol defines ways to establish pattern-like routing, where a queue will receive each message that was successfully matched against a regular expression. AMQP also defines even more advanced routing patterns, such as headers, providing even greater flexibility in network management. However, since the goal of SLAN is to provide basic primitives, direct matching against the key is sufficient.

The SLAN supports the following operations:

- ***Broadcast Operation:*** $f_{\text{broadcast}}(n_s, m)$ enqueues m into all $Q(n), n \in N$, where a broadcast from n_s via E_{fanout} delivers m to every node.
- ***Direct Send Operation:*** $f_{\text{sendDirect}}(n_s, n_t, m)$ enqueues m into $Q(n_t)$, where a direct send uses E_{direct} and $R(n_t)$ so that only n_t's queue receives m.
- ***Consumption Operation:*** $f_{\text{consume}}(n, m)$ such that if $(n, m) \in C$, then m is removed from $Q(n)$, where $C \subseteq N \times M$ is a consumption relation such that if $(n, m) \in C$, it indicates node n has consumed message m (i.e., removed it from its queue). Once a node n processes m, it's marked consumed and dequeued.

The simulated network could be perceived as an integral system that has its state and a set of internal processes that actively modify it. This approach allows us to gain a holistic understanding of the observed Decentralized Coordination Network (DCN) built on top of the underlying communication media.

State and its transitions could be defined in the following way:

- ***Broadcast Transition:*** $\delta_{\text{broadcast}}(S, n_s, m)$ adds m to all $Q(n)$. This transition signifies state change during the call of $f_{\text{broadcast}}(n_s, m)$.
- ***Direct Send Transition:*** $\delta_{\text{direct}}(S, n_s, n_t, m)$ adds m to $Q(n_t)$. This models the state change when a message is sent directly with $f_{\text{sendDirect}}(n_s, n_t, m)$.
- ***Consume Transition:*** $\delta_{\text{consume}}(S, n, m)$ removes m from $Q(n)$ and adds (n, m) to C, representing a state change when a node finishes processing a message.

The reliability management, congestion control, message persistence, and network tunneling are devolved to the SLAN layer. This abstraction allows us to simplify both the interface's complexity and functional concerns within RSDP, making it more digestible and robust. Further research could include finding different approaches towards establishing communication media between RSDP nodes appropriate for different environments.

The RSDP phases and interactions are shown in Fig. 2:

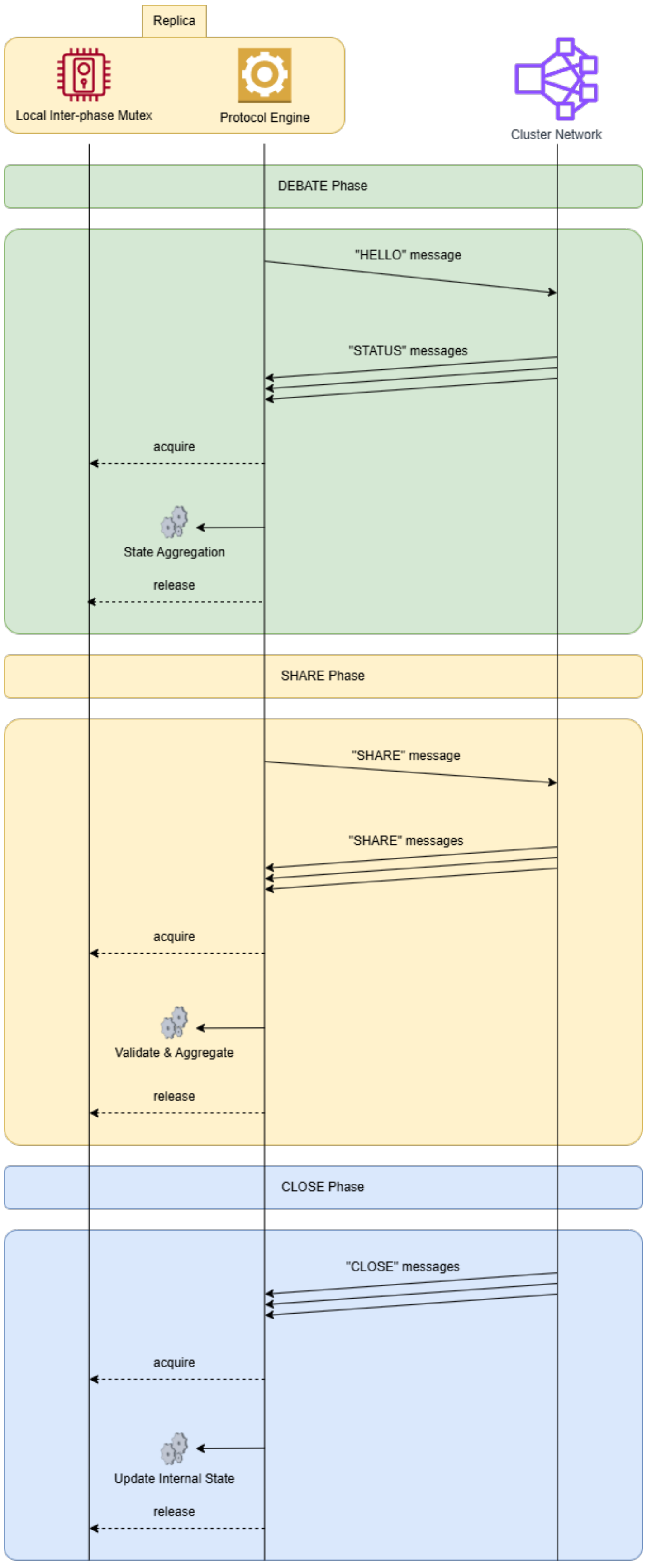

Figure 2: RSDP phases and interactions

The RSDP cluster could modeled as a directed graph $G_{\text{cluster}}=(V,E)$, where a set of replicas $V=\{v_1,v_2,\ldots,v_n\}$ and a set of directed edges $E\subseteq V\times V$ representing communication links between nodes form a topology of a distributed system. In such a system, each node has s_i which is the initial state of the node v_i and a dedicated mutex to ensure consistency between state transitions [15].

Let us first define utility functions such as node identifier: $f_{\text{id}}(v_i)\rightarrow \text{id of node } v_i$, derives the sender's address for a given node; metadata extraction $f_{\text{meta}}(v_i)\rightarrow \text{metadata of node } v_i$, derives the initial or meta information for a given node.

Every state mutation operation starts with $f_{\text{acquire}}(v_i)$ and ends with $f_{\text{release}}(v_i)$ function calls, to lock and release the local state transition mutex respectively. These procedures are necessary to guarantee that no two-state mutation functions such as f_{agg} or f_{update} could be executed at the same time and interfere with the results of each other.

The "DEBATE" phase includes the following steps:

- ***Send "HELLO" Messages:*** each v_i sends a "HELLO" message to its out-neighbors $N^{+(v_i)}$, where: $M_{\text{hello}}(v_i)=(f_{\text{id}}(v_i),f_{\text{meta}}(v_i))$ with a directed message propagation that could be denoted as: $M_{\text{hello}}(v_i)\xrightarrow{\text{propagated to}}N^{+(v_i)}$.
- ***Receive "STATUS" Messages:*** upon receiving M_{hello}, each $v_j\in N^{+(v_i)}$ sends its initial state: $M_{\text{status}}(v_j)=(f_{\text{id}}(v_j),f_{\text{meta}}(v_i),s_j)$.
- ***Aggregate States:*** each v_i aggregates the received states $\{s_j/v_j\in N^{-(v_i)}\}$ into a local aggregated state: $s_i^*=f_{\text{agg status}}(s_i,\{M_{\text{status}}(v_j)/v_j\in N^{-(v_i)}\})$

This process introduces replicas to each other within the system. Its purpose is to share the initial configuration or metadata to derive the initial view of the system's state.

The "SHARE" phase includes the following steps:

- ***Broadcast "SHARE" Messages:*** each v_i broadcasts its aggregated state s_i^* to its out-neighbors: $M_{\text{share}}(v_i)=(f_{\text{id}}(v_i),f_{\text{meta}}(v_i),s_i^*)$ with the directed message propagation as follows: $M_{\text{share}}(v_i)\xrightarrow{\text{propagated to}}N^{+(v_i)}$.
- ***Validate and Aggregate:*** each receiving node v_j validates and merges the received state using $\widehat{s_j^*}=f_{\text{agg share}}(s_j^*,\{M_{\text{share}}(v_i)/v_i\in N^{-(v_j)}\})$where $f_{\text{agg share}}$ extracts state components s_i^* from each $M_{\text{share}}(v_i)$ and performs operations declared within the reducer to derive a final operational state $\widehat{s_j^*}$.

After validation and aggregation processes are finished, the local state gets updated and represents a holistic view of the target cluster state. At this point, all the necessary steps were taken, and the participating nodes could execute their operations based on the derived states.

The "CLOSE" phase includes the following steps:

- ***Send "CLOSE" Messages:*** v_k sends a "CLOSE" message containing its state s_k to its out-neighbors: $M_{\text{close}}(v_k)=(f_{\text{id}}(v_k),f_{\text{meta}}(v_k),s_k)$ with directed message propagation denoted as: $M_{\text{close}}(v_k)\xrightarrow{\text{propagated to}}N^{+(v_k)}$.

- ***Update Internal State:*** each receiving node v_i removes references to v_k and updates its $s_i' = f_{\text{agg close}}(s_j, M_{\text{close}}(v_k))$ where $f_{\text{agg close}}$ is a function defined within the reducer to update the internal state s_i based on incoming records within s_k.

This phase is not necessary and is used to introduce dynamic participation handling capabilities for RSDP. The process allows to dynamically adjust the cluster state when a subset of nodes departs from the network.

3. Reinforcement of the phase transition control

Phase transition control is one of the primary aspects of the protocol's consistency guarantees. In its initial version, RSDP defined mutexes as a main instrument to isolate state manipulation logic between phases [15, 28]. Mutexes create so-called critical sections that restrict access to a single abstract execution entity. The approach allows for coordinating multiple such entities, viz., processes, their threads, or asynchronous threads in the case of event-driven architectures.

Nonetheless, the defined transition restrictions are not sufficient for handling outlying operations. Critical sections forbid parallel execution of the restricted code segments with programmatically defined boundaries. To enter the critical section and leave it, the parallel execution entity must acquire and release the mutex, respectively. If there are multiple entry points to the critical section and some of them omit the ingress boundary, parallelism and inconsistencies are still quite possible.

Having said that, RSDP in its original definition does not verify whether "STATUS" messages were received during the "DEBATE" phase or not. The controlling mutex by definition was taken before sending the initial "HELLO" message and released after the "SHARE" message was sent. That approach, while protecting against parallel processing of consistent sequential messages, allowed for some delayed messages to be reprocessed after the "DEBATE" phase was already finished.

To resolve this issue, a new state control mechanism is proposed leveraging the principles of the finite state machine (FSM) [29]. Let us first define a set of possible protocol execution statuses $G = \{g_{\text{INITIAL}}, g_{\text{DEBATE}}, g_{\text{IDLE}}, g_{\text{SHARE}}, g_{\text{CLOSE}}\}$, representing associated zones with the respective phases and inter-phase execution states.

In such a machine, transitions are defined as a set T, where each transition $t_n \in T$ is represented by: $t_n = (g_i, \sigma_n, g_j)$ where $g_i, g_j \in G$ and σ_n is the event that triggers the transition. The transition rules could be defined as follows:

- ***INITIAL → DEBATE:*** $t_1 = (g_{\text{INITIAL}}, \sigma_{\text{start}}, g_{\text{DEBATE}})$, triggered by a start signal from the joining node. The g_{INITIAL} status does not allow the execution of any operations besides the initiation of the protocol.
- ***DEBATE → IDLE:*** $t_2 = (g_{\text{DEBATE}}, \sigma_{\text{status agg}}, g_{\text{IDLE}})$ triggered when the aggregation from the "STATUS" messages is finalized. The g_{DEBATE} status does not allow execution of any operations besides those within "DEBATE" phase.
- ***IDLE → DEBATE:*** $t_3 = (g_{\text{IDLE}}, \sigma_{\text{hello}}, g_{\text{DEBATE}})$, triggered before sending the initial "HELLO" message during the "DEBATE" phase. The g_{IDLE} status does not allow execution of any operations besides transition to the next status.
- ***IDLE → SHARE:*** $t_4 = (g_{\text{IDLE}}, \sigma_{\text{share}}, g_{\text{SHARE}})$, triggered before sending or upon receiving the "SHARE" message. The g_{SHARE} status does not allow the execution of any operations besides those within "SHARE" phase.

- ***IDLE → CLOSE:*** $t_5 = \left(g_{\text{IDLE}}, \sigma_{\text{close}}, g_{\text{CLOSE}}\right)$, triggered upon receiving the "CLOSE" message. The g_{CLOSE} status does not allow the execution of any operations besides those within "CLOSE" phase.
- ***SHARE → IDLE:*** $t_6 = \left(g_{\text{SHARE}}, \sigma_{\text{share agg}}, g_{\text{IDLE}}\right)$ triggered when the aggregation from the "SHARE" messages is finalized. The g_{DEBATE} status does not allow the execution of any operations besides those within "SHARE" phase.
- ***CLOSE → IDLE:*** $t_7 = \left(g_{\text{CLOSE}}, \sigma_{\text{close agg}}, g_{\text{IDLE}}\right)$ triggered when the aggregation from the "CLOSE" messages is finalized. The g_{DEBATE} status does not allow the execution of any operations besides those within "CLOSE" phase.

Let $g_i \in G$ represent the current. The transition function δ is defined as: $\delta\left(g_i, \sigma\right) = g_j$ and is used to formally change the current execution status. For example, $\delta\left(g_{\text{INITIAL}}, \sigma_{\text{start}}\right) = g_{\text{DEBATE}}$.

Based on the defined rules, a state transition matrix could be constructed that visualizes available operations.

In the following matrix, states are indexed as follows:

1 → g_{INITIAL}

2 → g_{DEBATE}

3 → g_{IDLE}

4 → g_{SHARE}

5 → g_{CLOSE}

$$T_{\text{matrix}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{1}$$

The transition rules thereby forbid the execution of outlying operations and provide additional consistency guarantees for the protocol in addition to the already established mutex system. Before adding messages to the buffers or executing state mutating operations, the engine now also has to verify the validity of its status.

Having resolved the state transition inconsistencies, another key aspect in stochastic environments is to handle infrequent but still possible scenarios. One such scenario is an indefinite holding of a mutex due to unforeseen physical or logical interference. Since the protocol engine leverages such locks to modify the internal representation of the cluster state, it could potentially get stuck if for some reason the mutex was not released.

First, let us define mutex μ a binary variable where:

$$\mu = \begin{cases} 0 \wedge if\ unlocked \\ i \wedge if\ locked\ by\ t\,h\,read\ H_i \end{cases}$$

In that case, μ represents a mutex state and i is an identifier of the thread that locked the mutex. The mutex can have the following state transitions:

- ***Lock:*** $\mu = 0 \rightarrow \mu = i$ (mutex acquired by the thread i).
- ***Release:*** $\mu \leftarrow 0$ (mutex available to be acquired again).

Mutex allows isolating resource access between multiple remote cooperating parties that are trying to perform a critical operation simultaneously. In that context, a critical section is part of a program that ensures mutual exclusion to the shared resource:

$$\forall t, \sum_{i=1}^{n} \chi_i(t) \leq 1 \tag{2}$$

where t is some point in time, n is some threads, $\chi_i(t)$: characteristic function; $\chi_i(t)=1$ if H_i is in the critical section at time t, $\chi_i(t)=0$ otherwise.

To handle possible inconsistencies and deadlocks, a newer version of the protocol leverages the time-based mutexes. Essentially, this type of mutex enters a critical section with a commitment to release it within a certain time interval. A timeout-based mutex μ_t introduces a timer x_i and a timeout τ_i. The following state transitions and rules apply to the μ_t:

- ***Acquire Mutex:*** $\mu_t = 0 \qquad \mu_t \leftarrow i, x_i := 0$.
- ***Release Mutex:*** $\mu_t = i \wedge x_i < \tau_i \qquad \mu_t \leftarrow 0$.
- ***Timeout Exception:*** $\mu_t = i \wedge x_i \geq \tau_i \qquad \text{exception raised}, \mu_t \leftarrow 0$

Such a mutex does not resolve the underlying unexpected issues that led to the deadlock in the first place but allows us to detect the issues and react to them. The protocol engine could either go into recovery and try to reconcile the execution state or abort all operations and reinitialize itself, going through each synchronization phase again, starting with g_{INITIAL}.

4. Status consensus quorum introduction within RSDP

The initial descriptions of RSDP have left uncovered mechanisms of reducers' internal operation consistency. To begin a discussion about the importance of a quorum-based approach for a subset of distributed synchronization tasks, let us first define the failure and validity conditions for the protocol, its managed nodes, and the system as a whole.

The validity conditions could be evaluated from multiple perspectives and for different abstract objects. Firstly, there are perspectives of every node $v_i \in V$ within the system $G_{\text{cluster}} = (V, E)$ defined in the previous sections. The failure condition could also be evaluated from the perspective of the G_{cluster} itself based on different protocol stages.

Let us begin the definition of validity conditions from the perspective of participating nodes. The ***"Local Validity"*** implies the repeatable results of the aggregation methods. Formally said: v_i is locally valid if:

$$\forall t \in N, f_{\text{agg share}}\left(s_i^*, \{M_{\text{share}}(v_j) / v_j \in N^{-(v_i)}\}\right) = \widehat{s_j^*} \tag{3}$$

This means that repeated aggregator calls on the same data yield the same result. The validity implies determinism and that the aggregation function does not perform any unexpected side effects that would lead to desynchronization.

The other part of the validity conditions from the replica's perspective describes its relative operational quality in comparison with other participants. The ***"Global Validity"*** could be described formally as:

$$f_{\text{agg status}}\left(s_i, \{M_{\text{status}}(v_j) / v_j \in N^{-(v_i)}\}\right) = f_{\text{agg share}}\left(s_i^*, \{M_{\text{share}}(v_j) / v_j \in N^{-(v_i)}\}\right) \tag{4}$$

That could also be expressed as $s_i^* = \widehat{s_j^*}$, the locally aggregated view of the cluster state is representative and conforms with the holistic representation gained by merging the final states from participating nodes.

The perspective of a single node could be limited and thus biased. A simple binary validity condition is not indicative within the ambit of a distributed consensus-based system. Hence, the validation should be considered as a gauged estimation representing the degree of validity at some particular time point. The first validation condition from the system's perspective called

"Transitional System Validity" defines the divergence degree based on the last sharing session of aggregated states. The estimation process could be described in the following steps:

- ***Collect Broadcasted States:*** $S^* = s_i^* \subseteq M_{\text{share}}(v_i) / v_i \in V$;
- ***Group States:*** partition $S^{¿}$ by identical values into classes $\{P_1, \ldots, P_l\}$;
- ***Measure Divergence:*** apply one of the validity estimation functions defined below.

The measure could be performed by any participating node or external interceptor that has access to the communication media. The following base functions could be used for such estimation:

- ***Entropy:*** $D_{\text{en}} = -\sum_{l=1}^{m} \frac{|P_l|}{n} \log\left(\frac{|P_l|}{n}\right)$ where evaluation is based on entropy and a higher value means higher fragmentation [30].
- ***Concentration:*** $D_{\text{co}} = \sum_{l=1}^{m} \left(\frac{|P_l|}{n}\right)^2$, measure the squared distribution of elements, where a higher value means lover fragmentation.
- ***Magnitude:*** $D_{\text{mg}} = \frac{\underset{l}{max}|P_l|}{n}$, represents the normalized maximum proportion, where a higher value means lover fragmentation.

A similar approach could be leveraged to build the ***"Operational System Validity"*** which represents the divergence between the finalized states. The same steps for evaluation apply to the $\widehat{S_i^*} = \{\widehat{s_i^*} / v_i \in V, \widehat{s_i^*} \text{ belongs to } v_i\}$ being the initial set. The method requires access to the participating nodes to gather the finalized states.

Having discussed the validity conditions and their estimation, it is quite important to emphasize that the main metric representing the cluster's state is "Operational System Validity". That is because it describes the system's consistency rather than the individual node. Additionally, it is less susceptible to the biases of a single node. Thus, minimization of divergence is the primary goal of establishing a robust distributed consensus framework.

The protocol could leverage its redundancy to improve consistency. To achieve that, an additional layer is proposed to automatically reconcile inconsistent state aggregations. It introduces the popular and weighted voting mechanisms with a set of deterministic rules to be applied before the finalized state $\widehat{s_i^*}$ is settled for the replica [31, 32].

Let's start with the basis of the proposed voting consensus system. Suppose each node v_j proposes a hash $h(s_j^*)$, where s_j^* is the state aggregation of v_j. Then node v_i receives the states and hashes from $N^{-(v_i)} = \{v_j \in V / (v_j, v_i) \in E\}$, verifies them and selects one to represent its own finalized state $\widehat{s_i^*}$.

The ***"Unweighted Popular Vote"*** starts with the collection of $H_i = \{h(s_j^*) / v_j \in N^{-(v_i)}\}$, where each $h(s_j^*)$ represents the state aggregation from the perspective of a node v_j. To evaluate the vote result, for each candidate hash $h'_{i,k} \in H_i$, where $k \in N$ and $0 < k \leq |H_i|$ compute $C_{i,k}(h'_{i,k})$, which is the number of times $h'_{i,k}$ appears in H_i:

$$C_{i,k}(h'_{i,k}) = \left|\{h(s_j^*) \in H_i : h(s_j^*) = h'_{i,k}\}\right| \quad (5)$$

After each corresponding $C_{i,k}\left(h'_{i,k}\right)$ has been constructed, the vote decision could be expressed as follows:

$$\overline{C_{i,k}} = \max_{h'_{i,k} \in H_i} C_{i,k}\left(h'_{i,k}\right), T_i = \{h'_{i,k} \in H_i / C_{i,k}\left(h'_{i,k}\right) = \overline{C_{i,k}}\} \tag{6}$$

where T_i is a set of winning unique hash values. In such a case there could be a tie between votes that has to be resolved. If $|T_i| = 1$, choose that single hash h_i^* in T_i. Otherwise, pick $h_i^* = \min_{\prec} T_i$, where $\prec$ is a total order (e.g., lexicographical) on H_i.

Within the ***"Weighted Popular Vote"***, hash collection and h_i^* vote decision process is the same. What is different is the approach to implementing $C_{i,k}\left(h'_{i,k}\right)$ by introducing the weight term $w_j > 0$. For each node $v_j \in V$, let us define a weight $w_j \in R_0$. These weights can be derived in multiple different ways. For example, from trust relationships or resource capacities:

- ***Trust-Based Weights:*** $w_j = \sum_{v_l \in V} T\left(v_l, v_j\right), T\left(v_l, v_j\right) \geq 0$, where $T\left(v_l, v_j\right)$ measures how much v_l trusts v_j. Summing over all $v_l \in V$ yields a strictly positive real number if each node is trusted by at least one other node. The trust could belong to $R_{>0}$ or a simpler binary version {0, 1}.
- ***Resource-Based Weights:*** $w_j = f_{\mathrm{res}}\left(\mathrm{BW}_j, \mathrm{CPU}_j, \mathrm{RAM}_j\right)$, where $\mathrm{BW}_j, \mathrm{CPU}_j, \mathrm{RAM}_j \in R_{\geq 0}$ denote the bandwidth, CPU, and memory resources of v_j, and $f_{\mathrm{res}} : R^3_{\geq 0} \rightarrow R_{>0}$ is a function that returns a positive real number representing the node's capacity.

The weight function could be injected into the reducer, allowing for custom setups that adhere to the unique requirements of the system. It is important to emphasize that the weight assignment function should also be deterministic, otherwise it would negatively influence system state fragmentation. The weighted count computation function could be represented as the following:

$$C_{i,k}\left(h'_{i,k}\right) = \sum_{v_j \in N^{-(v_i)}} \left[w_j \cdot I\{h\left(s_j^*\right) = h'_{i,k}\}\right] \tag{7}$$

where $I\{X\}$ returns 1 if X is true and 0 otherwise. Each vote for $h'_{i,k}$ is multiplied by v_j's weight value w_j. The model could be generalized to include the popular voting mechanism where every vote has the same weight. Thus, the reducer's interface can simply expect a weight function where the default one would yield the same value for each node.

5. Enhancing protocol's efficiency and reliability

Moving on to the efficiency and reliability improvements, let us first consider the updates related to the critical section management within the protocol. Critical sections are the base atomic units of execution from the perspective of a parallelized system. These points are responsible for a unified state update shared between multiple execution threads. As we've discussed in the previous section, the critical sections are defined and managed by respective mutex instances responsible for ensuring phase transition consistency.

It is quite important to emphasize that while controlled critical sections allow to manage asynchronous access to the shared entity, they are one of the primary causes leading to bottlenecks and deadlocks in the systems. That is because critical sections practically merge multiple parallel execution threads into a single queue of operations. Therefore, they should be reduced to a minimal required space to guarantee successful code execution and, at the same time, avoid excessive and greedy locking strategies that would unnecessarily stagger the protocol's engine.

The proposed modification provides a new approach to delimit phase transitions and their internal operations. One of the most important is related to the "DEBATE" phase and its critical sections management. RSDP initially defined the said phase starting with the mutex acquisition before sending the "HELLO" message. Subsequent operations involved "STATUS" message gathering, state aggregation, and broadcasting the "SHARE" message. The interphase mutex was supposed to be released only after the broadcast operation, holding the entire system at a halt during the message-passing process. Since in its initial conception, the notion of engine statuses was not introduced, such an approach allowed to delay the processing of the incoming "SHARE" messages and avoid inconsistencies.

In contrast with the initial version of RSDP, the proposed modification suggests acquiring a mutex exclusively for the aggregation and update processes. The initial operations during the "DEBATE" phase involving cluster status gathering could be executed outside the critical section while still guaranteeing state transition consistency due to the introduction of the FSM principles that postulate strict ordering of engine status transitions and hence executable operations. Such an improvement allows us to avoid potential congestion related to superfluous critical sections.

Another aspect that must be addressed is the reasoning behind the "CLOSE" phase's existence and its purpose. This phase is responsible for gracefully and dynamically handling departing nodes from the cluster's set. It has to be said that the approach could be done differently without an introduction of the unique phase and its related logic. The same effect could be potentially achieved by using the "SHARE" message, which contains a modified state by the departing replica itself rather than the "CLOSE" message. Though the latter was eventually chosen due to the following reasons:

- The addition of a "CLOSE" message allows for differentiation between state updates and reduction requests. Separation of concerns greatly simplifies aggregation logic for the incoming shared states.
- Broadcasting "CLOSE" messages are more efficient than repetitively going through the entire "DEBATE" and "SHARE" phases, which would be necessary in case the reducer's logic is built on top of a newly proposed voting mechanism.
- The separated approach allows for the development of additional logic related to events management. An example could involve side effects that are not related to the state management.

Overall, the decision to include the "CLOSE" phase introduces greater modularity, simplifies the codebase, and extends potential logical extensions within the cluster. Any operation defined for "CLOSE" phase handling should still comply with deterministic principles to avoid desynchronization. If the reducer's logic relies on side effects during that phase, a new synch process starting with "DEBATE" will have to be executed.

The proposed RSDP 2.0 does not guarantee finalized state consistency. It is still quite possible for some replicas to diverge due to hardware, network, software conditions, or their combinations. Mechanisms like voting, including its weighted variants, aim to reduce the state groups' fragmentation degree, but they cannot guarantee eventual consistency.

For that reason, the proposed version introduces the ***"Mandatory Resynchronization Mechanism"***. The MRM could be implemented in multiple different ways, each with its implications:

- ***Asynchronous Scheduling:*** in that case, each replica would have an independent resynch period. The approach is straightforward but does not leverage the cluster's capabilities to coordinate its operations, leading to inefficient monitoring.

- ***Cron-Based Scheduling:*** this would lead to strict synchronization and result in near-simultaneous RSDP status transitions on participating nodes, but discrepancies detection would take the entire schedule period [33].
- ***Evenly Distributed Intervals:*** through RSDP itself, the participating replicas could distribute the time slot and coordinate with each other in intervals to monitor the cluster's state.

The latest approach would start with the initial synchronization round, going through the "DEBATE" and "SHARE" phases. Using the dedicated reducer, every replica could deduce the network size, participant, and deterministically assign the time slots for resynch periods. The approach minimizes the detection time of an inconsistency by leveraging the cluster's capabilities.

Recall that upon receiving M_{hello}, each $v_j \in N^{-(v_i)}$ sends its initial state: $M_{\text{status}}(v_j) = (f_{\text{id}}(v_j), f_{\text{meta}}(v_i), s_j)$ through the $M_{\text{status}}(v_j)$ message that between other things like metadata or state, contains a routable sender's address. That said, when the buffer of incoming messages is being processed into an aggregation $s_i^* = f_{\text{agg status}}(s_i, \{M_{\text{status}}(v_j) / v_j \in N^{-(v_i)}\})$, it could be constructed as follows:

$$s_i^* = \left\{ \left(f_{\text{id}}(v_j), \frac{\Delta T}{|A|} \cdot rank\left(f_{\text{id}}(v_j), A\right) \right) / v_j \in N^{-(v_i)} \cup v_i \right\} \tag{8}$$

where ΔT is a synchronization period provided as a parameter, $|A|$ is the total number of addresses in the sorted set A, where $A = \{f_{\text{id}}(v_j) / v_j \in N^{-(v_i)} \cup v_i\}$, and $rank\left(f_{\text{id}}(v_j), A\right)$ returns the position of $f_{\text{id}}(v_j)$ within a sorted set A for a given node v_j. As a result, it would represent the assigned timeslots for every participating node.

Another important issue that must be addressed pertains to the potential attacks aimed at service availability. Firstly, the designed version of RSDP is still supposed to be used within the private network, where each node goes through authentication and authorization processes to transmit messages inside the network. Though the attacks are not limited only to public services, in such cases, it is important to design countermeasures against the potential abuse of synchronization mechanisms.

Since every interaction process goes through the SLAN layer, which is comprised of a set of intermediary nodes, additional monitoring and security measures could be installed to prevent potential message spam abuse. This is possible since the media server is responsible for authentication and the routing process which enables dynamic evaluation of requests. If it detects frequent messages of the same type coming from the same node, it could easily isolate it to avoid potential network congestion and malicious activities.

Additionally, since the RSDP 2.0 introduces quorum-based consensus, the acceptance of an incoming shared state is resolved due to the BFT-compliant properties of the voting mechanism. That is, for a new shared state to be accepted, it should be supported by the majority of the network. That means that an incoming divergent state would lead to a new phase of the synchronization period that would determine whether the transition should be accepted. Future research could be aimed at detecting anomalous behavior.

6. Decentralized byzantine fault tolerance

As was previously stated, RSDP is designed for controlled environments and relies on the SLAN layer for authentication and security measures. By leveraging a voting-based census, it is possible to limit the influence on the state distribution and coordination process since any malicious intent would be dismissed unless it comes from the majority of the network.

Nonetheless, RSDP is designed as a cluster solution that would coordinate a group of nodes towards a common goal through the common state management capabilities. The protocol does not provide any side-effect verification mechanism and thus cannot guarantee that the designated tasks through state mapping were accomplished correctly. Additionally, it relies on participating nodes to provide status and aggregated data without malicious intent since every other node would trust it. For these reasons, to achieve decentralization and leverage the protocol's capabilities outside the controlled environment, blockchain technology could be leveraged [16–20].

The RSDP-Blockchain layer interactions are shown in Fig. 3:

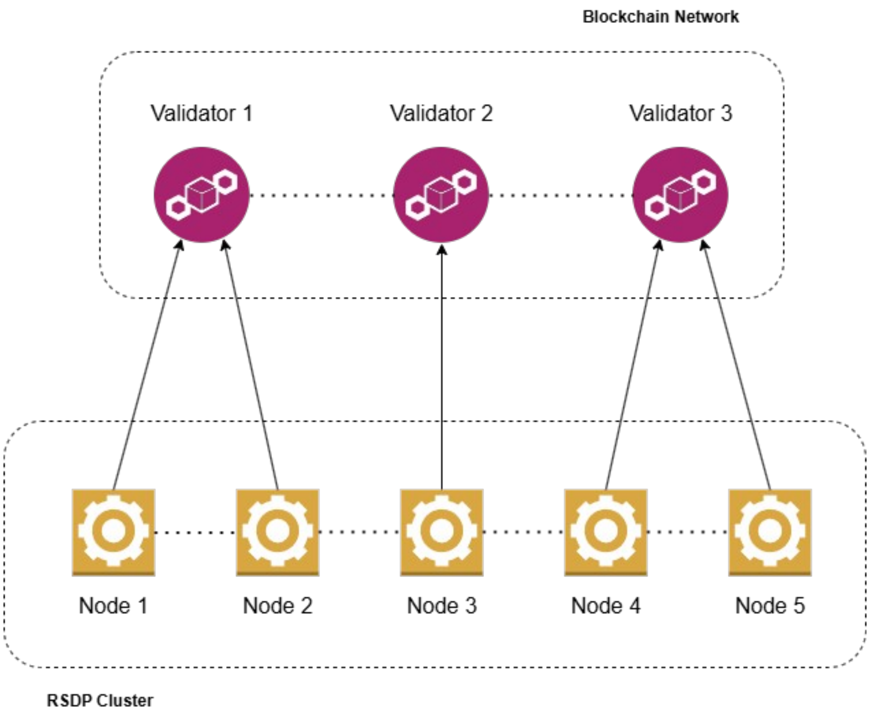

Figure 3: RSDP-Blockchain layers

As shown in Fig. 3, there are two distinct layers defined, the first being the blockchain network and its validator nodes and the other RSDP cluster nodes. Every node within the RSDP cluster has to have connections established with each other as well as with the validator node serving as an entry point for interactions with the ledger. To explain the reasoning behind such a combination, let us first discuss blockchain technology, its implications, limitations, and the problems it allows us to solve.

Blockchain is a decentralized network of publicly linked nodes responsible for managing the ledger's history. In its foundation, it relies on the linked list of hashes, representing a succinct state of the assigned block. One of the core problems that is solved with blockchain is the historical consensus regarding occurring interactions. Its operational consistency is guaranteed by a common consensus mechanism and strict validation rules [16–20].

There are myriads of approaches to establishing a voting mechanism, with the most common being proof of stake, proof of authority, and proof of work. Their purpose is to achieve network-wide consensus about the next block value. In contrast with RSDP, we can define a clear distinction between the blockchain consensus layer and the designed protocol:

- RSDP is designed as a solution that provides distributed state management and coordination, where every node contributes to the resulting aggregation while blockchain consensus protocols are aimed towards the validation process to verify the next block in the ledger.

- Blockchain consensus protocols are designed to scale; they do not require a fully connected network for their operation, which is a current limitation of RSDP and will be addressed in future research.
- RSDP emphasizes distribution rather than decentralization. The proposed mechanisms allow for mitigating the influence of uncertain network conditions, but the purpose of the protocol is to coordinate a cluster rather than provide a trustless platform.

That being said, blockchain technology could serve as a trust provision layer, responsible for asset management and slashing penalties in case state inconsistency is detected. Every node within the RSDP cluster should be registered on-chain, and a prearranged amount of assets should be staked for both potential penalties and rewards. On top of that, every node must commit its state transition events to the chain for historical audit purposes.

Every interaction with the cluster would follow the following stages:

- A set of RSDP nodes register on the blockchain network, providing stakes to the designated smart contract.
- The task being published on the ledger through the contract would signify the beginning of cluster processing. The task could include execution parameters, target goals to be achieved, and the chosen state reducer.
- Participating nodes register a cluster through the smart contract and sign up for task execution.
- Moving through every state transition phase such as “HELLO”, “SHARE”, or “CLOSE”, logs would be published on the ledger to verify operational correctness.
- Once the task is finalized due to either an end condition or a revoking command by the client, the results and latest states are to be published on the ledger for verification.

The approach allows following the log trace at any point since the ledger is a publicly available structure. Recall that every state transition function within RSDP should be deterministic, and thus, every operation could be verified by outside entities. In case inconsistency was detected on the node during its phases, slashing could be applied based on the impact degree, which could be determined by methods provided in previous sections such as “Operational System Validity” based on concentration, entropy, or the largest group.

Another concern that has to be addressed is the state mutation due to external information. It could happen that the cluster must agree upon a value that has a significant impact on other systems. For example, RSDP could be used for cluster-wide rate limits compliance, where every node tries to access a remote server and must avoid security policy violations. In such cases, it is quite common to have dynamic policies that adjust to the current demand. Hence, the state should periodically be updated to reflect the current demand.

Inside the controlled environment, it's possible to simply establish an additional communication channel or leverage existing SLAN’s capabilities to broadcast a new event that would update the system’s status and trigger a new “DEBATE” round. The said approach would not be applicable within the public environment due to trust limitations and accountability. Hence, the ledger must be utilized as a trustless event propagation medium that would record the originating address and provide economic guarantees for participants.

Conclusions

Modern requirements towards available computational capabilities, their coordination, and security inevitably lead to the rising academic and engineering interest of the networking community. As a result, a plethora of methods of achieving dynamic coordinated network extension and maintenance have been developed, where RSDP stands as a unique approach for abstracting complexities related to distributed consensus management.

Firstly, this article expounds on the definition of the updated RSDP model, including its protocol phases and consensus process. In addition to that, it introduces new methods aimed at safeguarding the phase transition process described within the protocol. This version of the protocol also defines additional mechanisms for handling lost updates and flooding conditions that may happen due to hardware issues or intentionally. The proposed approach allows for a significant improvement in the protocol's resilience and reliability.

Secondly, this article introduces a new BFT-compliant approach with RSDP by defining a set of quorum state reducers. The quorum coordination allows for effectively discovering malicious operations during consensus, both intentional and accidental, which is a primary property required from shared distributed systems. The designed foundation for quorum support is flexible and easy to modify due to the open definition of the weight control mechanism.

Finally, this article defines a new computational coordination approach by incorporating both RSDP and blockchain technologies. Such design allows for an efficient, rapid, and incentivized collaboration of interlinked nodes within a decentralized system. Blockchain technology in that case works as both an incentive and governance platform, providing additional resources for the correct cooperation with the nodes and slashing procedures otherwise. Future applications of this technology could lead to an expansion of accessible, low-cost cloud computation engines suitable for various tasks.

Overall, the tenet of this article is to further expansion of the possibility horizon within a decentralized network coordination paradigm. This paper is intended to ameliorate decision-making processes done by network engineers and architects. It is also a goal of this article to spark a further surge in research and engineering efforts within decentralized coordination technology.

Declaration on Generative AI

While preparing this work, the authors used the AI programs Grammarly Pro to correct text grammar and Strike Plagiarism to search for possible plagiarism. After using this tool, the authors reviewed and edited the content as needed and took full responsibility for the publication's content.

References

[1] V. Grechaninov, et al., Models and Methods for Determining Application Performance Estimates in Distributed Structures, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3288, 2022, 134–141.

[2] V. Astapenya, et al., Analysis of ways and methods of increasing the availability of information in distributed information systems, in: 2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (2021). doi:10.1109/picst54195.2021.9772161

[3] H. Hulak, et al., Dynamic model of guarantee capacity and cyber security management in the critical automated system, in: 2nd International Conference on Conflict Management in Global Information Networks, vol. 3530 (2023) 102–111.

[4] R. Guerraoui, A. Schiper, Fault-tolerance by replication in distributed systems, in: Reliable Software Technologies—Ada-Europe'96, Lecture Notes in Computer Science, vol. 1088, 1996. doi:10.1007/BFb0013477

[5] K. P. Birman, T. A. Joseph, Exploiting replication in distributed systems, Distrib. Syst. (1989) 319–367. doi:10.1145/90417.90751

[6] B. Ciciani, et al., Analysis of replication in distributed database systems, IEEE Trans. Knowl. Data Eng. 2 (1990) 247–261. doi:10.1109/69.54723

[7] M. Zaharia, et al., Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in: 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI '12), 2012, 1–14.

[8] F. Cristian, Understanding fault-tolerant distributed systems, Commun. of the ACM 34(2) (1991) 56–78. doi:10.1145/102792.102801

[9] A. Luntovskyy, et al., Highly-distributed systems: What is inside? in: 2020 IEEE International Conference on Problems of Infocommunications. Science and Technology (PIC S&T), 2020. doi:10.1109/PICST51311.2020.9467890

[10] V. S. Pai et al., Locality-aware request distribution in cluster-based servers, ACM SIGOPS OSR 32(5) (1998) 205–216. doi:10.1145/384265.291048

[11] A. Verma, et al., Large-scale cluster management at Google with Borg, in: 10th European Conference on Computer Systems, EuroSys'15, 2015, 1–17. doi:10.1145/2741948.2741964

[12] Y. Kostiuk, et al., Integrated protection strategies and adaptive resource distribution for secure video streaming over a Bluetooth network, in: Cybersecurity Providing in Information and Telecommunication Systems II, vol. 3826, 2024, 129–138.

[13] P. Anakhov, et al., Evaluation method of the physical compatibility of equipment in a hybrid information transmission network, J. Theor. Appl. Inf. Technol. 100(22) (2022) 6635–6644.

[14] M. Kotov, S. Toliupa, V. Nakonechnyi, Method of building local area network simulation based on AMQP and its support protocols suite, Telecommun. Inf. Technol. 3 (2024) 102–119. doi:10.31673/2412-4338.2024.039989

[15] M. Kotov, S. Toliupa, V. Nakonechnyi, Replica state discovery protocol based on advanced message queuing protocol, Electron. Prof. Sci. J. Cybersecur. Educ. Sci. Tech. 3(23) (2024) 156–171. doi:10.28925/2663-4023.2024.23.156171

[16] Z. Zheng, et al., An overview of blockchain technology: Architecture, consensus, and future trends, in: IEEE International Congress on Big Data (BigData Congress), 2017. doi:10.1109/BigDataCongress.2017.85

[17] D. Yaga, et al., Blockchain technology overview, National Institute of Standards and Technology Internal Report, 2019. doi:10.48550/arXiv.1906.11078

[18] J. Golosova, A. Romanovs, The advantages and disadvantages of the blockchain technology, in: IEEE 6th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE), 2018. doi:10.1109/AIEEE.2018.8592253

[19] G. Habib, et al., Blockchain technology: Benefits, challenges, applications, and integration of blockchain technology with cloud computing, Future Internet 14(11) (2022). doi:10.3390/fi14110341

[20] M. S. Kotov, Tree-based state sharding for scalability and load balancing in multichain systems, Electron. Prof. Sci. J. Cybersecur. Educ. Sci. Tech. 2(26) (2024) 392–408. doi:10.28925/2663-4023.2024.26.702

[21] W. Liu, et al., Distributed and parallel blockchain: Towards a multi-chain system with enhanced security, IEEE Transact. Dependable Secur. Comput. 22(1) (2024) 1–16. doi:10.1109/tdsc.2024.3417531

[22] S. I. Sion, et al., A comprehensive review of multi-chain architecture for blockchain integration in organizations, in: Business Process Management: Blockchain, Robotic Process Automation, Central and Eastern European, Educators and Industry Forum, BPM 2024, Lecture Notes in Business Information Processing, vol. 527, 2024. doi:10.1007/978-3-031-70445-1_1

[23] F. Hashim, K. Shuaib, N. Zaki, Sharding for scalable blockchain networks, SN Comput. Sci. 4(2) (2023). doi:10.1007/s42979-022-01435-z

[24] V. Zhebka, et al., Methodology for choosing a consensus algorithm for blockchain technology, in: Workshop on Digital Economy Concepts and Technologies Workshop, DECaT, vol. 3665 (2024) 106–113.

[25] N. Naik, Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP, in: 2017 IEEE International Systems Engineering Symposium (ISSE), 2017, 426–435. doi:10.1109/SysEng.2017.8088251

[26] J. L. Fernandes, et al., Performance evaluation of RESTful web services and AMQP protocol, in: 2013 5th International Conference on Ubiquitous and Future Networks (ICUFN), 2013. doi:10.1109/ICUFN.2013.6614932

[27] N. Q. Uy, V. H. Nam, A comparison of AMQP and MQTT protocols for Internet of Things, in: 2019 6th NAFOSTED Conference on Information and Computer Science (NICS), 2019. doi:10.1109/NICS48868.2019.9023812

[28] A. Holub, The mutex and lock management, Taming Java Threads, 2000, 61–79. doi:10.1007/978-1-4302-1129-7_3

[29] M. Ben-Ari, F. Mondada, Finite State Machines, Elements of Robotics, 2018, 55–61. doi:10.1007/978-3-319-62533-1_4

[30] P. Saraiva, On Shannon entropy and its applications, Kuwait J. Sci. 50(3) (2023) 194–199. doi:10.1016/j.kjs.2023.05.004

[31] S. Leonardos, D. Reijsbergen, G. Piliouras, Weighted voting on the blockchain: Improving consensus in proof of stake protocols, Int. J. Netw. Manag. 30(5) (2020) e2093. doi:10.1002/nem.2093

[32] F. Benezit, P. Thiran, M. Vetterli, The distributed multiple voting problem, IEEE J. Selected Topics Signal Proces. 5(4) (2011) 791–804. doi:10.1109/JSTSP.2011.2114326

[33] Scheduling Tasks, Beginning Fedora, Apress, 2007, 427–431. doi:10.1007/978-1-4302-0297-4_32