



DOI 10.28925/2663-4023.2026.35.1443

UDC 004.72:004.75

Maksym Kotov

PhD student at the department of cyber security and information protection

Taras Shevchenko National University of Kyiv, Kyiv, Ukraine

ORCID: 0000-0003-1153-3198

maksym_kotov@ukr.net

Serhii Toliupa

Doct. Sci. (Engineering), Professor,

Professor of the department of cyber security and information protection

Taras Shevchenko National University of Kyiv, Kyiv, Ukraine

ORCID: 0000-0002-1919-9174

serhii.toliupa@knu.ua

HANDLING SPLIT BRAIN SCENARIOS WITHIN REPLICA STATE DISCOVERY PROTOCOL

Abstract. Distributed systems often require coordination capabilities for resource and task allocation among participating nodes. Traditional coordination solutions rely on centralized control planes or leader election algorithms involving risks of authoritative decisions. In that regard, the Replica State Discovery Protocol (RSDP) was created to achieve similar capability in a one-rank network. RSDP defines peer-to-peer communication phases aimed at gathering a cluster state consisting of participants' perspectives. The architecture takes into account diverse requirements of different execution environments with a stratified system and modular approach towards constructing cluster state reducers responsible for their own data slices. When a network must organize into a coherent system, the potential losses of messages and nodes, as well as entire communication links, must be considered and handled on the protocol's design layer. One of the common pitfalls in that regard is network fragmentation and the requirement to guarantee cluster stability in such scenarios. Previous work of RSDP has been concentrated on crash fault tolerance and operational stability but has also explicitly acknowledged the absence of a formal mechanism for handling split-brain events. The purpose of this article is to provide a theoretical basis for building state reducers of two categories: those that tolerate network partitioning and those requiring additional reconciliation mechanisms. It is also within the ambit of this work to formally describe and verify such a mechanism without extending the initial set of messages used for the protocol and avoiding additional communication or coordination overhead to handle such events.

Keywords: distributed computing; crash fault tolerance; failure detectors; state synchronization; cluster management; state convergence; Replica State Discovery Protocol (RSDP); eventual consistency.

INTRODUCTION

The Replica State Discovery Protocol was created as a solution for the general-purpose coordination problem in distributed systems without reliance on an authoritative plane [1-4]. It is built upon the principles of layered abstraction and architecture, which allows it to isolate complexities from spilling into adjacent domains. RSDP is also built on the principle of modularity, which is mostly visible in its state reducer architecture. The reducers could take disjoint responsibility for different state slices while using the same coordination primitives and synchronization rounds. By extending respective layers, RSDP could be adapted to distributed and decentralized environments [2-3].

Previous work in the series described the crash fault tolerance mechanism of RSDP and provided proofs that the TTL eviction mechanism is an eventually perfect failure detector [5]. The work also provided closed-form bounds for convergence which were later tested in both sandboxed and real environments. In Theorem 3 of that article, the network partition event was addressed, describing what happens before the partition and after and how the reconciliation works, but the work explicitly limits the mechanics during the fragmentation and declares it as a scope of future work [5]. There could be different types of reducers that imply different requirements toward handling such events. For example, those requiring exactly one answer, such as leader election, imply that during partition, the mechanism has to prevent multiple simultaneous leaders from converging in subnetworks.



To establish bounds of what could be achieved from this work, we should first mention the work of Gilbert and Lynch [6], which formalized Brewer's conjecture and showed that a partitioned system cannot at the same time guarantee availability and single-copy consistency. Later work [7] provided insight that the previous result could be a design choice made per partition. RSDP works within a partially synchronous model of Dwork, Lynch, and Stockmeyer [8], with a failure detection mechanism based on TTL state eviction as described in Chandra and Toueg [9], and thus avoids FLP impossibility [10]. In this work we chose to work with Brewer's framing, meaning that the decision about the CAP trade-off could be done on a per-reducer basis. To do that we would have to formally describe both types of the reducers and the constraints they impose on their members.

The first class of the reducers, which does not require coordination, uses basis and structures that were studied in works of Shapiro et al. [11], which showed that a replicated state forming join-semilattice does not require additional coordination. Hellerstein and Alvaro [12] described the CALM theorem, saying that a distributed module could be consistent if it is monotone. In the previous work, RSDP already admitted CRDTs as possible state reducer implementations, but it did not clearly separate monotone reducers from others.

As for the coordinated side, it could also be done in a few ways: the model described in Isis [14] allows only one partition to make progress, while Extended Virtual Synchrony [15], the specification of Babaoğlu, Davoli, and Montresor [16], and the systems described by Chockler, Keidar, and Vitenberg [17] allow merging partial views during the merge event. To avoid critical point of failure on the primary partition, leases [18] could be used as described in Chubby [19] and Raft [20]. Another important mechanism to avoid stale updates is epoch numbering, as was used in viewstamped replication [21]. Weighed voting could be used to govern the decision-making process and provide more influence for more credible nodes [22]. There are also established mechanisms and research about handling overlapping majorities without making a few primary partitions [20, 23]. To handle catastrophic or planned reloads of nodes, incarnation numbers could be used like in SWIM to distinguish between participation phases [24].

Having said that, the purpose of this article is not in creating novel techniques of split-brain handling or discovering new coordination primitives. The goal is to adapt the existing mechanism as shown to be used in RSDP for its purposes. Self-merging data structures, leases for holding primary references, epoch numbers, and incarnations should be adapted in the context of RSDP to handle two primary classes of reducers: those requiring coordination under a partitioned view of the network and those that do not need any additional mechanisms beyond reducers conforming to properties declared in previous works [5]. It is therefore the goal of this article to also minimize mechanical changes in RSDP to achieve the sad functionality and to leverage existing mechanisms such as TTL eviction and data exchange to achieve the desired effect.

EXTENDED MODEL

To describe a process of handling split brain events, in this section, we introduce additional mechanics for the RSDP. First is electorate attached to the "SHARE" payload with an epoch and an additional parameter θ_r , which is an additional timeout parameter shorter than eviction threshold. That mechanic will be used to make a decision whether the node is authoritative and can actually derive cluster state. Second, an epoch for electorate to handle shadow nodes and reappearing peers.

First, let's start with the voter configurations $\mathcal{K} = (\varepsilon, \Lambda, w)$, where $\varepsilon \in \mathbb{N}$ is an epoch; electorate set is expressed as $\Lambda \subseteq \mathcal{V}$; and $w: \Lambda \rightarrow \mathbb{R}_+$ are weights assigned to the electorate members. Every node v_i in the cluster holds its own $\mathcal{K}_i(t)$ to make the state derivation decisions. When a node freshly enters the cluster or rejoins it after crashing it gets an initial empty state $\mathcal{K}_i(t) = \perp = (0, \emptyset, \emptyset)$ or a preconfigured set of nodes through administrative channels. With that, the "SHARE" payload changes and includes the state derivation data itself, node incarnation number from a previously defined model [5], and the voter configurations supplied as the third param in the tuple $P_i^+ = (P_i, \iota_i, \mathcal{K}_i)$. Then upon delivery of a payload with voter configs $\mathcal{K}' = (\varepsilon', \Lambda', w')$:

$$\mathcal{K}_i^t = \mathcal{K}' \text{ if } \varepsilon' > \varepsilon_i^{t-}, \text{ else } \mathcal{K}_i^{t-}. \quad (1)$$

We define the voting wait for a subset of cluster nodes $S \subseteq \mathcal{V}$ as a sum of all node weight in the electoral configurations that are both in S and Λ :

$$W_{\mathcal{K}}(S) = \sum_{j \in S \cap \Lambda} w(j), \quad (2)$$

where nodes that are not a part of the electoral configurations set (where not injected into configuration set by an administrator) do not have any voting power.

We denote $Maj_{\mathcal{K}}(S)$ as a majority voting group in given configurations set if majority condition holds:

$$Maj_{\mathcal{K}}(S) \Leftrightarrow W_{\mathcal{K}}(S) > \frac{W_{\mathcal{K}}(\Lambda)}{2}, \quad (3)$$

with the majority margin:

$$\mu(\mathcal{K}) = \min \left\{ W_{\mathcal{K}}(X) - \frac{W_{\mathcal{K}}(\Lambda)}{2} : X \subseteq \Lambda, Maj_{\mathcal{K}}(X) \right\}. \quad (4)$$



Next, adjacent configuration $\mathcal{K}' = (\varepsilon', \Lambda', w')$ is the next step after adding or removing a node from the electorate $\mathcal{K} = (\varepsilon, \Lambda, w)$. For a configuration to be considered adjacent, the following conditions must hold: (a) epoch should be an increment $\varepsilon' = \varepsilon + 1$; (b) the electorates should differ by at most one node $|\Lambda \oplus \Lambda'| \leq 1$, where $\oplus$ shows the difference between sets; (c) weights of other nodes did not change $w' = w$, on $\Lambda \cap \Lambda'$; (d) and the increment of weight added by a new node is at most $2\mu(\mathcal{K}_-)$, where $\mathcal{K}_-$ is configuration without the new node.

The adjacency is used to validate the configuration set updates issued by the administrator. Let us define the injection operation $inj_i(\mathcal{K}')$ which means that a configurations set $\mathcal{K}'$ should be installed at node v_i , and is only accepted in case if the node v_i holds authority in that moment (has the majority quorum) and at the same time, the $\mathcal{K}'$ is an adjacent configuration to $\mathcal{K}_-$.

The implication here is that an electoral pool of nodes is a set of stable nodes that could be significantly smaller than the entire set of cluster nodes. Hence, the scaling operations could be done independent of the current electoral set and would not require administrative intermissions. The point of this mechanism is that in a dynamic cluster, an electoral set has to be stable, otherwise it could end up with an authoritative threshold weight of 5 when having 20 nodes which could lead to the split-brain scenarios.

Lemma 1 (adjacent majorities intersect). If $\mathcal{K}$ and $\mathcal{K}'$ satisfy requirement from the definition of an adjacent configuration, then every majority in $\mathcal{K}$ intersects very majority in $\mathcal{K}'$.

Proof. If two electoral configuration sets are equal and the update only increments the epoch, then having two disjoint majorities would imply that their weight would be bigger then total sum of weights. If not, then let S be a set without a new node e and $T = S \cup \{e\}$, and majority of S as X and the majority of T as Y such that $X \cap Y = \emptyset$. With that $Y \setminus \{e\} \subseteq S \setminus X$, meaning $W(Y) \leq W(S) - W(X) + w(e)$ while $W(Y) > \frac{W(S)+w(e)}{2}$, hence $W(X) < \frac{W(S)}{2} + \frac{w(e)}{2}$ and $W(X)$ would then have a margin less than $\frac{w(e)}{2}$, which contradict requirement (d) from definition of an adjacent configuration set.

With $0 < \theta_r < \theta$ and quarantine $Q > 0$ (a minimum tenure a node has to hold for its weight to be counted), freshness, voting, and authentication mechanisms apply to figure out if node is authoritative (can produce a state):

$$Fresh_i(t) = \{j \in dom \Sigma_i(t) : C_i(t) - \tau_i^t(j) < \theta_r\} \cup \{i\}, \quad (5)$$

where $\Sigma_i(t)$ is the current view of state collection gathered from “SHARE” messages, $C_i(t)$ is the local clock of the node v_i and $\tau_i^{t(j)}$ is when it was last heard from.

$$Vote_i(t) = \{j \in Fresh_i(t) \cap \Lambda_i(t) : C_i(t) - \kappa_i^t(j) \geq Q\}, \quad (6)$$

gives a condition for the voting count, where only admin-approved electoral candidates have weight. Here, $\kappa_i^t(j)$ is when i first saw j 's current incarnation and $C_i(t) - \kappa_i^t(j)$ is how long it was known.

$$Auth_i(t) = \mathbb{I} \left\{ W_{\mathcal{K}_i}(Vote_i(t)) > \frac{W_{\mathcal{K}_i}(\Lambda_i(t))}{2} \right\}. \quad (7)$$

gives a condition of authoritative state of a node. All nodes inside RSDP cluster would still provide perspectives that will be used for state derivation, authoritative status just says to a particular node if it can trust its own derivation logic and can produce a state. That means that a translator would drop state $\perp$ whatever the incoming perspectives from other nodes are if it does not have authority.

Lemma 2 (handover gap). Given $0 < \theta_r < \theta$ and $\frac{\theta}{1+\bar{\rho}} > \Delta + \frac{\theta_r + T_{sweep} + k \cdot T_{resync}}{1-\bar{\rho}}$, and if partition happens at $t_0 \geq GST$, then every node in minority segment will have $Auth=0$ before they get evicted from any other node through TTL mechanism.

Proof. Any message that could cross the partition boundary would arrive at $t_0 + \Delta$, where Δ is message delay bound [5]. Since nothing will arrive after that bound, a peer would stop being fresh at $\frac{\theta_r}{1-\bar{\rho}}$, and $Auth$ would be reevaluated at $\frac{T_{sweep}}{1-\bar{\rho}}$ and yield 0 since $Vote$ won't hold a majority. Also, since at most $k-1$ messages could be lost and sent at intervals T_{resync} the node will hear it after $t_0 - k \cdot \frac{T_{resync}}{1-\bar{\rho}}$, and eviction will apply after $\frac{\theta}{1+\bar{\rho}}$.

Lemma 3 (quarantine). With $\frac{Q}{1+\bar{\rho}} \geq \Delta + \frac{\theta_r + T_{sweep}}{1-\bar{\rho}}$, if the node restarts at any point, it will not count on other nodes before its previous incarnation will get removed from the view set of currently running nodes.

The argument here repeats from Lemma 2, with restart time as a reference point. Since the rejoining node must wait the quarantine time, its previous incarnation will be evicted to avoid double counting of the same entity and potential split-brain events.

Every node must also retain $\hat{t}_i(j)$, an incarnation number seen from all its peers. Then if the node v_j reappears and $t_j = \hat{t}_i(j)$, it means it was missing because of partitioning, and in other cases, if $t_j > \hat{t}_i(j)$, it means that the node has restarted.



PARTITION-AWARE ADMISSIBILITY

In RSDP, state reducers is a set of functions working one by one in a predefined pipeline schema [1]. The main components in the reducer defining its nature and purpose are the aggregation and translation functions. Starting with aggregation function G , which takes a batch b , which is the multiset of payloads from space $\mathcal{X}$ taken from the node memory Σ_i , along with the previous internal state $s \in \mathcal{S}$, and then produces a new state. Then, a translator function T takes the aggregated state and transforms it into a useful view for the client. For example, aggregator could perform membership gathering, putting identifiers into a set, while translator could give a count of members and ordinal position for the client.

One of the results of the previous research aiming to prove crash fault tolerance of RSDP was a formal admissibility ruleset for a state reducer that guarantees that when incoming data is the same – the nodes will derive the same states in the cluster. The limitations require reducer chain to be deterministic, independent of order permutations, independent of duplications, and have historical independence. The problem here is that partitioned networks will not have the same set of data to aggregate on nodes, which could create inconsistencies. That's why in this section we are introducing a new property.

To compare outputs of different reducers, each declares its fact map F , showing exactly the list of facts, the client will act upon after obtaining the state view through translation function. For cluster members reducers with the resulting set $\{a, b\}$, the fact map would contain two claims: “a is a member” and “b is a member”. For leader reducer a fact map would have a single claim “node a is the leader”.

With that, a new property Monotonicity (M) which delimiters partition-tolerant reducers from those depending on additional consistency mechanisms:

$$b \subseteq b' \Rightarrow F(V(b)) \subseteq F(V(b')), \quad (8)$$

which is basically saying that the reducer is partition-tolerable if it cannot retract its claim because of additional data. For example, with members reducers, if the cluster heals, the other part held node c, the reducer will not retract memberships of nodes a and b. In contrast, for leadership reducer, the appearance of node c could change the claim of the reducer, depending on the algorithm. The principle of monotonic reducer based on what it returns comes from monotonicity of the CALM theorem [12].

Proposition 1 (sufficient condition). Let $\sqcup$ be an operation that is associative, commutative, idempotent operation on $\mathcal{S}$, and let us also define relation $x \sqsubseteq y$ which is equivalent to $x \sqcup y = y$. Equation 8 holds if given batches b and b' , if $G(b \sqcup b', s) = G(b, s) \sqcup G(b', s)$ and $F \circ T$ maps $\sqsubseteq$ -larger states to $\subseteq$ -larger fact sets, where $\sqcup$ is a multiset union, and $\circ$ shows function composition $(F \circ T)(x) = F(T(x))$.

Proof. For batches $b \subseteq b'$, $G(b') = G(b) \sqcup G(b' \setminus b)$, hence $G(b) \sqsubseteq G(b')$, and $F \circ T$ carries $\sqsubseteq$ to $\subseteq$.

State-based CRDTs were studied in many works and have been proven to have both properties of crash-fault-tolerant admissibility of reducers, as well as monotonicity property described in equation 8 [11, 13].

Proposition 2 (merge on heal). Given a reducer that is partition-safe and given a partition event that splits network into $P_1, \dots, P_m$ with its own state views V_k , which after remerging of network will produce V , $F(V_k) \subseteq F(V)$ for every k .

Proof. As was shown in crash fault tolerance analysis, after remerging networks, each node will hold in its memory Σ_i a superset of states of all partitioned networks. Hence, b_k satisfies $b_k \subseteq b$ and equation 8 will apply.

THEOREMS

Having established two types of possible reducers in context of split-brain handling, this section will concentrate on proving properties about partition-sensible reducers. The first theorem pertains to the guarantee that partitions without authority will not produce conflicting state reducer results, while the second theorem aims to prove that legitimately authoritative partitions will continue functioning under such events.

The results work under a few following conditions:

1. Partitioning happens at time $t_0 \geq GST$, where network splits into a set of subnetworks inside which nodes can communicate between each other but not across the boundary, and there is also an authoritative subnet which has the majority of nodes derived from electorate before partitioning.
2. At t_0 the cluster is converged and participating nodes agree on electorate set and configuration set $\mathcal{K}$.
3. Until the network remerges and heals from partitioning, nodes do not crash, restart, change membership status. This particular condition is only for the first theorem and is relaxed for the second.
4. Parameters of RSDP engine satisfy conditions from Lemma 2 and 3, and the lower bound of θ_r .
5. Administrative configuration changes happen, as per initial assumption, once an epoch.

Then, by Lemma 2, subnetworks, and consequently nodes that are not in the authoritative network will stop producing a state to the client for a set of partition-sensitive reducers in the following bound:



$$t_d = t_0 + \Delta + \frac{\theta_r + T_{sweep}}{1 - \rho}, \quad (9)$$

and that also means no node will evict partitioned peers from its state and view before the specified bound t_d .

Theorem 1 (no split-brain). After partitioning event at t_0 and reconvergence bound t_d , only a single authoritative subnetwork will produce state outputs for partition-sensitive reducers, while others will yield $\perp$. During the transitive period, all subnetworks will still produce the same result for partition-sensitive reducers.

Proof. Before t_d nodes will still hold the same electorate set and state outputs that was obtained during previous synchronization session, including conditions 2 and 3 from the list above, all non- $\perp$ values will be the same. After t_d , following Lemma 2, only authoritative subnetwork with the majority of voters will produce a state; the electorate set cannot change more than once per epoch as per assumption; and two disjoint subnetworks can't both represent the majority and gain authority at the same time.

Theorem 2 (authority where possible). Throughout the partitioning event, the majority subnetwork holds authority and continues to produce states for the clients. After the network remerges and partitioning heals at t_h , all participating nodes from the majority subnetwork as well as other partitions will converge in the following bounds:

$$\max\left\{T_{conv}, T_{hb} + \frac{Q + T_{sweep} + D_{max}}{1 - \rho}\right\}. \quad (10)$$

Proof. Electorate stay fresh within θ_r , majority subnetwork will not lose its authority and will reconverge in T_{conv} , while evicting partitioned nodes treating them as crashed ones. After healing, peers receive “SHARE” from lost nodes within T_{hb} , and the electorate restores within Q , authority reevaluated in $T_{sweep}/(1 - \rho)$, and state $D_{max}/(1 - \rho)$.

TLA+. The mechanized component for TLA specification was written and executed with TLC version 2.19. All the specifications, as well as the execution logs and parameters are specified in the GitHub repository attached in source links of this article [25]. Protocol operates over time but to simplify and reduce the state space, in this specification we discretize the time in ticks, where each tick SHARE message is being sent. At most $k-1$ messages could be lost and SHARE message could be delivered in at most 1 tick. The parameters used by default are $\Theta = 5$, $\theta_r = 2$, $Q = 4$, $k = 2$ and $\Delta = 1$. Checked: D – NoDualLeader, C – ConsistentOutputs, A – MajorityKeepsAuth, R – AuthReturns, M – MergeOnHeal, L – Lemma 1.

Table 1

TLA specification evaluation results

Configuration	n	Mechanism removed	Checked	Result	Distinct states	Depth	Time
lemma1 nocap / cap	—	weight cap	L	X / ✓	1 / 1	1 / 1	3s / 1s
gate_off	3	authority gate	D	X	973	7	1s
agg_fresh	3	aggregation over Σ	D	X	196	4	< 1s
order_off	3	$\theta_r < \Theta$ ($\theta_r = \Theta$)	D	X	985	7	1s
quar off / on	3	quarantine ($Q = 0$)	D, C	X / ✓	5,605 / 10,984	9 / 16	2s / 3s
tenure evict / fresh	3	tenure reset on staleness	D, C, R	X / ✓	4,400 / 20,968	8 / 10	2s / 15s
adj off / on	4	adjacency	D, C	X / ✓	6,463 / 1,278	8 / 7	1s / 1s
spacing off / on	4	spacing (A3)	D, C	X / ✓	10,871 / 18,166	9 / 16	3s / 7s
cap off / on	4	weight cap	D, C	X / ✓	1,567 / 318	8 / 7	< 1s
tie	4	— (stale change in a tie)	D, C	✓	9,722	9	4s
merge_n3	3	—	D, M, R	✓	20,968	10	15s
rec_tight	3	— (bound $k + Q - 1$)	R	X	453,371	9	59s
full_n3	3	— (any split, crash, restart, heal)	D, C, A, R	✓	1,045,001,965	≤ 38	4h 06min

The execution results showed that excluding any of the introduced mechanisms leads to a duplicate leader situation in a span of a few states. Additionally, during execution another failure mode was found related to the



tenure mechanism. When split brain scenario happens and then heals itself but one portion still receives delayed SHARE message in 1 tick, 2 leaders could emerge because the smaller partition does not consider freshness for the tenure. Thus, another requirement for tenure is to be continuously fresh.

CONCLUSIONS

Organising distributed systems into a coherent service entails complexities related to unstable transmission channels, participants, and the environment. Hence, it is important for the coordination protocols to provide robust operational mechanisms and guarantees of operational stability during their execution in the face of message loss or network fragmentation. In that light, RSDP has been proven already to handle message losses, participant departures, and, within this article, also the network fragmentation scenarios.

Within this work, a new property (L) lattice-mergeability of the reducer has been defined to differentiate between partition-tolerant components and those requiring the fragmentation-handling mechanism. This allows us to also establish a network where the overhead of sending an explicit electorate list could be omitted in favour of performance.

Secondly, this article provides a formal definition of the mechanism required to guarantee the mutual exclusion of authority, single authoritative component, and liveness of the coordination cluster. Multiple scenarios of network fragmentation have been considered, including latent node participations and rejoins. Properties of such networks were verified through the use of TLA specifications.

Overall, the purpose of this article was to address the consistency gap in previous versions of the RSDP during network fragmentation scenarios. With that confirmed, RSDP benefits from a formally proven operational stability in the face of both crash faults and network partitions, making it a reliable tool for non-adversarial environments. Addressing the Byzantine mechanism is a scope of future work.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Kotov, M., & Toliupa, S. (2025). Layered architecture for RSDP V3.0: Modular distributed consensus and coordination. In *Networks and sustainability. TCSET 2024* (Lecture Notes in Electrical Engineering, Vol. 1473). Springer. https://doi.org/10.1007/978-3-032-02272-1_6
2. Kotov, M., et al. (2025). Byzantine fault tolerance in distributed systems: Advancing the Replica State Discovery Protocol v2.0. *CEUR Workshop Proceedings*, 3991, 121–138. <https://ceur-ws.org/Vol-3991/paper10.pdf>
3. Kotov, M., Toliupa, S., & Buchyk, O. (2026). Decentralization model for the Replica State Discovery Protocol. In *2026 IEEE 18th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)* (pp. 1–6). IEEE. <https://doi.org/10.1109/TCSET65181.2026.11461147>
4. Toliupa, S., et al. (2024). Stateful cluster leader failover models and methods based on Replica State Discovery Protocol. *CEUR Workshop Proceedings*, 3933, 120–140. https://ceur-ws.org/Vol-3933/Paper_10.pdf
5. Kotov, M., & Toliupa, S. (2026). *Crash fault tolerance in the Replica State Discovery Protocol: Formal convergence guarantees and empirical validation* [Manuscript submitted for publication].
6. Gilbert, S., & Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2), 51–59. <https://doi.org/10.1145/564585.564601>
7. Brewer, E. (2012). CAP twelve years later: How the “rules” have changed. *Computer*, 45(2), 23–29. <https://doi.org/10.1109/MC.2012.37>
8. Dwork, C., Lynch, N., & Stockmeyer, L. (1988). Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2), 288–323. <https://doi.org/10.1145/42282.42283>
9. Chandra, T. D., & Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2), 225–267. <https://doi.org/10.1145/226643.226647>
10. Fischer, M. J., Lynch, N. A., & Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2), 374–382. <https://doi.org/10.1145/3149.214121>
11. Shapiro, M., et al. (2011). Conflict-free replicated data types. In *Stabilization, safety, and security of distributed systems* (Lecture Notes in Computer Science, Vol. 6976, pp. 386–400). Springer. https://doi.org/10.1007/978-3-642-24550-3_29
12. Hellerstein, J. M., & Alvaro, P. (2020). Keeping CALM: When distributed consistency is easy. *Communications of the ACM*, 63(9), 72–81. <https://doi.org/10.1145/3369736>
13. Almeida, P. S. (2023). *Approaches to conflict-free replicated data types* [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.2310.18220>



14. Birman, K., & Joseph, T. (1987). Exploiting virtual synchrony in distributed systems. In *Proceedings of the 11th ACM Symposium on Operating Systems Principles (SOSP '87)* (pp. 123–138). Association for Computing Machinery. <https://doi.org/10.1145/41457.37515>
15. Moser, L. E., et al. (1994). Extended virtual synchrony. In *Proceedings of the 14th International Conference on Distributed Computing Systems (ICDCS)* (pp. 56–65). IEEE. <https://doi.org/10.1109/ICDCS.1994.302392>
16. Babaoğlu, Ö., Davoli, R., & Montresor, A. (2001). Group communication in partitionable systems: Specification and algorithms. *IEEE Transactions on Software Engineering*, 27(4), 308–336. <https://doi.org/10.1109/32.917522>
17. Chockler, G. V., Keidar, I., & Vitenberg, R. (2001). Group communication specifications: A comprehensive study. *ACM Computing Surveys*, 33(4), 427–469. <https://doi.org/10.1145/503112.503113>
18. Gray, C., & Cheriton, D. (1989). Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles (SOSP '89)* (pp. 202–210). Association for Computing Machinery. <https://doi.org/10.1145/74850.74870>
19. Burrows, M. (2006). The Chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI '06)* (pp. 335–350). USENIX Association. <https://dl.acm.org/citation.cfm?id=1298487>
20. Ongaro, D., & Ousterhout, J. (2014). In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)* (pp. 305–319). USENIX Association. <https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro>
21. Oki, B. M., & Liskov, B. H. (1988). Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proceedings of the 7th ACM Symposium on Principles of Distributed Computing (PODC '88)* (pp. 8–17). Association for Computing Machinery. <https://doi.org/10.1145/62546.62549>
22. Gifford, D. K. (1979). Weighted voting for replicated data. In *Proceedings of the 7th ACM Symposium on Operating Systems Principles (SOSP '79)* (pp. 150–162). Association for Computing Machinery. <https://doi.org/10.1145/800215.806583>
23. Lamport, L., Malkhi, D., & Zhou, L. (2010). Reconfiguring a state machine. *ACM SIGACT News*, 41(1), 63–73. <https://doi.org/10.1145/1753171.1753191>
24. Das, A., Gupta, I., & Motivala, A. (2002). SWIM: Scalable weakly-consistent infection-style process group membership protocol. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2002)* (pp. 303–312). IEEE. <https://doi.org/10.1109/DSN.2002.1028914>
25. Replica State Discovery Protocol. (n.d.). *Fault-tolerance-tla: TLA+ specification and TLC verification artifacts* [Source code]. GitHub. Retrieved September 13, 2026, from <https://github.com/Replica-State-Discovery-Protocol/fault-tolerance-tla>

**Котов Максим Сергійович**

аспірант кафедри кібербезпеки та захисту інформації

Київський національний університет імені Тараса Шевченка, Київ, Україна

ORCID: 0000-0003-1153-3198

maksym_kotov@ukr.net

Толіупа Сергій Васильович

д.т.н., професор, професор кафедри кібербезпеки та захисту інформації

Київський національний університет імені Тараса Шевченка, Київ, Україна

ORCID: 0000-0002-1919-9174

serhii.toliupa@knu.ua

ОБРОБКА СЦЕНАРІЇВ РОЗДІЛЕННЯ МЕРЕЖІ В РАМКАХ ПРОТОКОЛУ ВИЯВЛЕННЯ СТАНУ РЕПЛІКИ

Анотація. Розподілені системи часто потребують засобів координації для розподілу ресурсів і завдань між вузлами-учасниками. Традиційні рішення для координації спираються на централізовані площини керування або алгоритми вибору лідера, що пов'язано з ризиками, властивими прийняттю рішень єдиним авторитетним вузлом. У цьому контексті було розроблено протокол виявлення стану реплік (RSDP) для забезпечення аналогічних можливостей у мережі рівноправних вузлів. RSDP визначає етапи взаємодії між вузлами, спрямовані на формування стану кластера на основі даних, отриманих від окремих учасників. Архітектура враховує вимоги різних середовищ виконання завдяки багаторівневій системі та модульному підходу до створення компонентів зведення стану (редукторів), кожен з яких відповідає за певний фрагмент даних. Коли мережа має організуватися в цілісну систему, на рівні проєктування протоколу необхідно передбачити обробку ситуацій, пов'язаних із втратою повідомлень, виходом із ладу вузлів або розривом каналів зв'язку. Однією з поширених проблем у цьому аспекті є фрагментація мережі та необхідність гарантування стабільності кластера за таких умов. Попередні дослідження RSDP були зосереджені на стійкості до відмов типу «зупинка» (crash faults) та операційній стабільності, проте в них також визнавалася відсутність формального механізму обробки ситуацій розколу системи (split-brain). Мета цієї статті – створити теоретичне підґрунтя для побудови редукторів стану двох категорій: тих, що стійкі до розділення мережі, і тих, що потребують додаткових механізмів узгодження даних. Також у межах цієї роботи здійснюється формальний опис і верифікація такого механізму без розширення початкового набору повідомлень протоколу та без додаткових витрат ресурсів на комунікацію чи координацію для обробки зазначених ситуацій.

Ключові слова: розподілені обчислення; відмовостійкість до збоїв; детектори збоїв; синхронізація станів; управління кластерами; конвергенція станів; протокол виявлення стану репліки (RSDP); остаточна узгодженість.

СПИСКИ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kotov, M., & Toliupa, S. (2025). Layered architecture for RSDP V3.0: Modular distributed consensus and coordination. In *Networks and sustainability. TCSET 2024* (Lecture Notes in Electrical Engineering, Vol. 1473). Springer. https://doi.org/10.1007/978-3-032-02272-1_6
2. Kotov, M., et al. (2025). Byzantine fault tolerance in distributed systems: Advancing the Replica State Discovery Protocol v2.0. *CEUR Workshop Proceedings*, 3991, 121–138. <https://ceur-ws.org/Vol-3991/paper10.pdf>
3. Kotov, M., Toliupa, S., & Buchyk, O. (2026). Decentralization model for the Replica State Discovery Protocol. In *2026 IEEE 18th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)* (pp. 1–6). IEEE. <https://doi.org/10.1109/TCSET65181.2026.11461147>
4. Toliupa, S., et al. (2024). Stateful cluster leader failover models and methods based on Replica State Discovery Protocol. *CEUR Workshop Proceedings*, 3933, 120–140. https://ceur-ws.org/Vol-3933/Paper_10.pdf
5. Kotov, M., & Toliupa, S. (2026). *Crash fault tolerance in the Replica State Discovery Protocol: Formal convergence guarantees and empirical validation* [Manuscript submitted for publication].



6. Gilbert, S., & Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2), 51–59. <https://doi.org/10.1145/564585.564601>
7. Brewer, E. (2012). CAP twelve years later: How the “rules” have changed. *Computer*, 45(2), 23–29. <https://doi.org/10.1109/MC.2012.37>
8. Dwork, C., Lynch, N., & Stockmeyer, L. (1988). Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2), 288–323. <https://doi.org/10.1145/42282.42283>
9. Chandra, T. D., & Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2), 225–267. <https://doi.org/10.1145/226643.226647>
10. Fischer, M. J., Lynch, N. A., & Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2), 374–382. <https://doi.org/10.1145/3149.214121>
11. Shapiro, M., et al. (2011). Conflict-free replicated data types. In *Stabilization, safety, and security of distributed systems* (Lecture Notes in Computer Science, Vol. 6976, pp. 386–400). Springer. https://doi.org/10.1007/978-3-642-24550-3_29
12. Hellerstein, J. M., & Alvaro, P. (2020). Keeping CALM: When distributed consistency is easy. *Communications of the ACM*, 63(9), 72–81. <https://doi.org/10.1145/3369736>
13. Almeida, P. S. (2023). *Approaches to conflict-free replicated data types* [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.2310.18220>
14. Birman, K., & Joseph, T. (1987). Exploiting virtual synchrony in distributed systems. In *Proceedings of the 11th ACM Symposium on Operating Systems Principles (SOSP '87)* (pp. 123–138). Association for Computing Machinery. <https://doi.org/10.1145/41457.37515>
15. Moser, L. E., et al. (1994). Extended virtual synchrony. In *Proceedings of the 14th International Conference on Distributed Computing Systems (ICDCS)* (pp. 56–65). IEEE. <https://doi.org/10.1109/ICDCS.1994.302392>
16. Babaoğlu, Ö., Davoli, R., & Montresor, A. (2001). Group communication in partitionable systems: Specification and algorithms. *IEEE Transactions on Software Engineering*, 27(4), 308–336. <https://doi.org/10.1109/32.917522>
17. Chockler, G. V., Keidar, I., & Vitenberg, R. (2001). Group communication specifications: A comprehensive study. *ACM Computing Surveys*, 33(4), 427–469. <https://doi.org/10.1145/503112.503113>
18. Gray, C., & Cheriton, D. (1989). Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles (SOSP '89)* (pp. 202–210). Association for Computing Machinery. <https://doi.org/10.1145/74850.74870>
19. Burrows, M. (2006). The Chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI '06)* (pp. 335–350). USENIX Association. <https://dl.acm.org/citation.cfm?id=1298487>
20. Ongaro, D., & Ousterhout, J. (2014). In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)* (pp. 305–319). USENIX Association.
21. Oki, B. M., & Liskov, B. H. (1988). Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proceedings of the 7th ACM Symposium on Principles of Distributed Computing (PODC '88)* (pp. 8–17). Association for Computing Machinery. <https://doi.org/10.1145/62546.62549>
22. Gifford, D. K. (1979). Weighted voting for replicated data. In *Proceedings of the 7th ACM Symposium on Operating Systems Principles (SOSP '79)* (pp. 150–162). Association for Computing Machinery. <https://doi.org/10.1145/800215.806583>
23. Lamport, L., Malkhi, D., & Zhou, L. (2010). Reconfiguring a state machine. *ACM SIGACT News*, 41(1), 63–73. <https://doi.org/10.1145/1753171.1753191>
24. Das, A., Gupta, I., & Motivala, A. (2002). SWIM: Scalable weakly-consistent infection-style process group membership protocol. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2002)* (pp. 303–312). IEEE. <https://doi.org/10.1109/DSN.2002.1028914>
25. Replica State Discovery Protocol. (n.d.). *Fault-tolerance-tla: TLA+ specification and TLC verification artifacts* [Source code]. GitHub. Retrieved September 13, 2026.

Отримано редакцією журналу / Received: 24.08.26

Прорецензовано / Revised: 30.08.26

Схвалено до друку / Accepted: 26.10.26



This work is licensed under Creative Commons Attribution-noncommercial-sharealike 4.0 International License.