Date of publication DEC-31, 2024, date of current version NOV-07, 2024.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.23.4.3756

Adaptive Consensus Algorithms: Designing for Durability against Unstable Network Connections

STANISLAV ZHURAVEL, OLHA SHPUR, MYKHAILO KLYMASH
Department of Telecommunication, Lviv Polytechnic National University, Lviv, 79013, Ukraine

Corresponding author: Olha Shpur (e-mail: olha.m.shpur@lpnu.ua).

ABSTRACT In distributed systems, achieving a consensus among nodes is crucial for ensuring data integrity and operational synchronization. A prevalent obstacle in this context is the instability of network connections, which can significantly undermine system performance and reliability. This article delves into a sophisticated strategy for refining consensus algorithms, aiming to introduce adaptability and fortify resilience against the unpredictability of network conditions. It describes and proposes a new method that modifies traditional consensus mechanisms to better withstand the challenges posed by unstable network environments. The essence of the method is to solve the consensus problem by dynamically adjusting the network parameters to match the real-time connection better. Further analysis of the system operation during the time of correct functioning allows us to detect failures with the help of a timeout, which signals the loss of communication with a node with which it is not possible to exchange messages. This approach makes it possible to improve the system's conclusion about the malfunction of a particular node and avoid possible false conclusions about its malfunction. Adjusting the delay value can help maintain stable system performance under variable network conditions.

KEYWORDS consensus algorithms; network instability; fault tolerance; simulation model; distributed systems.

I. INTRODUCTION

IN the dynamic landscape of distributed computing, the ability of nodes within a network to reach consensus is fundamental to the integrity and efficiency of distributed systems. These systems underpin a variety of critical applications, from blockchain technologies to distributed databases, where the consistency of shared data is paramount. However, the inherent challenge of maintaining this consistency becomes pronounced in the face of unstable network connections – a common and yet significantly impactful issue that can lead to data inconsistencies, decreased system performance, and reduced reliability [1, 2, 15].

The susceptibility of distributed systems to network instabilities calls for innovative solutions that can enhance the resilience of consensus mechanisms. Traditional consensus algorithms, while effective under stable conditions, often fall short when confronted with the erratic nature of real-world network environments. This limitation underscores the need for a new approach that not only recognizes the variable nature of network connectivity but also adapts to it, ensuring uninterrupted system performance and data integrity.

This article addresses this pressing need by introducing a novel modification to existing consensus algorithms, aimed at bolstering their robustness against network instability. By reimagining consensus mechanisms with a focus on adaptability, the proposed method offers a promising solution to one of the most persistent challenges in distributed computing. The essence of this approach lies in its dynamic adjustment of consensus parameters in response to the fluctuating conditions of the network, thereby maintaining a consistent and reliable consensus process.

In presenting this innovative method, the article aims to contribute to the broader discourse on distributed system design, offering a theoretical foundation that can guide future research and development. The introduction of this adaptive consensus algorithm not only marks a significant step forward in the quest for more resilient distributed systems but also sets the stage for empirical investigations that will further elucidate its practical implications and potential benefits.

A. PROBLEM STATEMENT

In the intricate ecosystem of distributed systems, where consensus algorithms are fundamental in maintaining the coherence and reliability of the network, unstable connections pose a significant threat. The operational efficiency and the very essence of distributed consensus, that is, the ability to achieve agreement on a single data value among nodes is at risk when faced with the erratic nature of network connectivity. Instances of latency variability and temporary disconnections can precipitate a cascade of operational challenges, including increased transaction times, throughput bottlenecks, and, most critically, the potential for the loss of consensus.

Achieving consensus in clusters involves various methods, each designed to ensure data consistency and reliability across distributed nodes. The primary classifications of consensus methods include leader-based, leaderless, quorum-based, Byzantine fault-tolerant (BFT) [3-5], proof-based, hierarchical, and hybrid approaches. Leader-based methods, such as Raft and Paxos, rely on electing a leader to coordinate decisions and manage the replication of data. Leaderless methods, like gossip protocols and some blockchain systems, do not have a central leader, instead relying on decentralized coordination among nodes. Quorum-based methods use subsets of nodes to approve decisions, ensuring consistency by requiring agreement from a majority or quorum. Byzantine fault-tolerant algorithms, such as Practical Byzantine Fault Tolerance (PBFT), are designed to withstand nodes that may behave maliciously or arbitrarily. Proof-based methods, like Proof of Work (PoW) and Proof of Stake (PoS) [6-10], are commonly used in blockchain systems to achieve consensus without centralized control. Hierarchical methods, such as LEACH, focus on energy efficiency by organizing nodes into clusters with designated leaders. Hybrid methods combine elements from different consensus approaches to leverage their strengths and mitigate weaknesses, such as Algorand's combination of cryptographic sortition with Byzantine agreement.

Among these consensus methods, leader-based protocols like Raft [11], Paxos (specifically Multi-Paxos), and ZooKeeper's Zab protocol [12-15] incorporate the notion of a leader and rely on delays to detect failures. In these systems, a leader is elected to coordinate actions and ensure consistency, while followers use timeouts to detect the absence of leader heartbeats, which indicates potential failures. If a leader fails to send heartbeats within a specified timeout period, followers assume the leader is unreachable and initiate a new election. This timeout mechanism effectively uses delays to maintain system availability and quickly adapt to changing conditions. By leveraging leader election and delay detection, these protocols focus on ensuring robust and efficient operation in distributed environments.

The central issue that this article addresses is the vulnerability of leader-based consensus algorithms to network instability, which can be ubiquitous in real-world applications due to various factors, such as hardware failures, bandwidth fluctuations, and dynamic network topologies [16-19]. Despite the robust design of consensus protocols like Raft, the reality of network unpredictability necessitates a better understanding of its impact on system performance and the critical thresholds where system efficiency begins to degrade significantly.

This work aims to dissect the problem of constant leader reelection induced by network delays in distributed systems running consensus algorithms like Raft. This condition is not merely a performance concern but a systemic threat that can disrupt the entire consensus process, leading to a state of confusion within the cluster and undermining the fidelity of the distributed ledger or database.

The problematic nature of this issue is multifaceted:

- the sensitivity of consensus algorithms to the precise timing of heartbeat messages for leader election and the consequential risk of frequent, unnecessary reconfigurations in the face of network instability;
- the need for a delicate balance in timeout settings to differentiate between actual leader failures and transient network delays, which is not well-defined in the face of varying network conditions;
- the implications of frequent leader re-elections, which include not only performance degradation but also the heightened risk of 'split-brain' scenarios and data inconsistencies.

In addressing these challenges, the article proposes to explore new methods and modifications to consensus algorithms that bolster resilience against unstable network conditions. The goal is to offer a novel approach that can maintain the integrity and consistency of distributed systems, thereby ensuring their operational efficiency and reliability in an unpredictable network landscape.

II. SELECTING THE BEST LEADER IN DISTRIBUTED NETWORK

Before implementing any modifications to consensus algorithms, it is crucial to discern the optimal configuration for peak system performance when running such protocols.

In the domain of leader-based consensus algorithms, such as Raft, the ideal scenario is straightforward: the most robust cluster is one where the leader maintains the most reliable network connections with its followers. This implies minimal delay between the leader and its followers. Recalling the operational principles of the Raft algorithm, it becomes apparent that Raft is not designed to identify the most optimal leader within the cluster; instead, its primary objective is to manage system failures and ensure the cluster's collective operation, without specific consideration for efficiency.

Nevertheless, algorithms like Raft are not inherently equipped to address real-world challenges, such as those mentioned previously, where network instabilities lead to a "frozen" state of the cluster due to constant leader reelection triggered by a few problematic network links.

In the domain of distributed computing, the election of an optimal leader within a cluster is paramount for ensuring efficient data management and system functionality. This necessitates a comprehensive analysis of attributes and conditions conducive to the selection of the most suitable leader among the nodes [20-22].

In the discourse on distributed systems, particularly those reliant on consensus algorithms, the issue of identifying the most optimal leader at any given juncture emerges as pivotal [23-25]. This inquiry is inherently temporal, contingent upon the prevailing network stability, which is subject to fluctuations over time. Consequently, an assertion regarding the superior suitability of a node as a leader can only be accurately made within the context of a specific temporal snapshot, given the rapid evolution of network conditions.

The table below shows potential network technologies where a cluster running a consensus algorithm can be deployed, along with possible instability factors that could significantly impact network performance at specific point in time and, consequently, the execution of the consensus algorithm.

Table 1. Network technologies and instability factors

Technology	Description	Instability factors
MPLS/IP	Efficient routing for enterprise networks	Latency variability, bandwidth fluctuations
BLE	Low-power, short-range wireless technology	Interference, limited range
WSN	Networks of spatially distributed sensors	Environmental interference, energy constraints
5G	High-speed mobile network technology	Cell handover issues, network congestion
SAT-Links	Satellite communication links for remote areas	Weather interference, signal delay
Mesh	Networks with multiple data paths for robustness	Dynamic topology changes, node failures

Latency in network technologies can vary significantly based on the use case, environmental factors, and infrastructure. While transmission latency might be low for high-speed links, other factors like propagation and processing latencies can contribute significantly to the overall delay. Understanding these aspects helps in designing better communication infrastructures and distributed systems.

To elucidate this concept, a comparative analysis of two nodes within a hypothetical network cluster, designated as node 4 (the incumbent leader) and node 1 (a potential candidate for leadership), is presented in Figure 1.

Assuming node 1 is aiming for leadership, determining its suitability compared to node 4 involves assessing the network latency to all other nodes within the cluster. This process is simplified to a comparative analysis of the delays encountered in communications originating from both nodes to their peers across the network. This approach underscores the fluidity of leadership efficacy in distributed systems, where the optimal leader is not a static designation but rather a reflection of the network current operational landscape. Through this lens, methodologies for leader selection that are both adaptive and reflective of the underlying network dynamics can be discerned.

The optimal leader is thus determined not solely by its status or capabilities but by the aggregate efficiency it brings to the network communication.

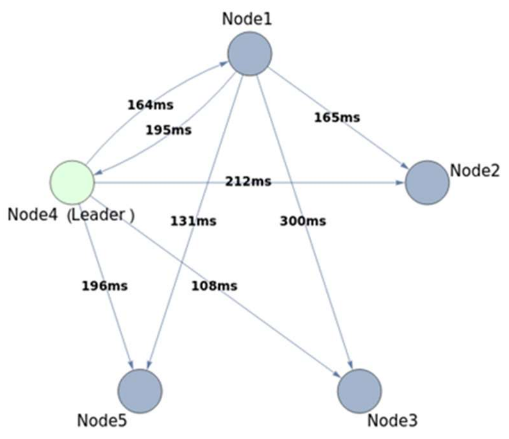


Figure 1. Cluster latency map for leader election

To mathematically formalize this notion, one could denote $L(1, j)$ as the latency between node i and node j. For a given node 1 vying for leadership, its total latency T_1 can be expressed as the sum of latencies between node 1 and all other $n-1$ nodes in the network:

$$T_1 = \sum_{j=2}^{n} L(1, j). \tag{1}$$

Similarly, the current leader node 4 has a corresponding total latency T_4:

$$T_4 = \sum_{j=1, j\neq 4}^{n} L(4, j). \tag{2}$$

In this context, node 1 would be a more suitable leader if T_1 is less than T_4, indicating that node 1 has a lower overall latency in communicating with the rest of the network:

$$Node\_1\_is\_more\_suitable\_if\ \_T_1 < T_4. \tag{3}$$

According to the methodology, the node with the lower total latency is more suitable leader. Comparing T_1 and T_4 is as follows:

$$T_1(835ms) > T_4(676ms). \tag{4}$$

Thus, node 4, with a total latency of 676ms, is more suitable as a leader than node 1 with a total latency of 835ms, based on the given configuration.

This evaluation assumes that overall lower latency correlates directly with better performance and suitability for leadership. However, this simplistic methodology overlooks scenarios wherein a single connection experiences significantly higher latency than the others, which plays a critical role in the comprehensive evaluation.

Let us enhance the reliability of connections from Node 1 to other nodes, except one link that exhibits significantly higher latency (refer to Figure 2).

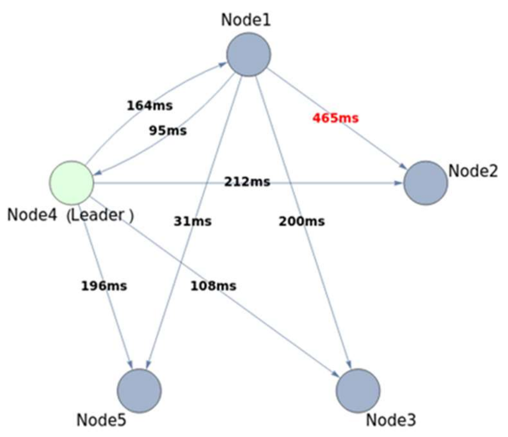


Figure 2. Cluster latency map with anomalous link

Utilizing the methodology above to identify the most suitable leader might initially suggest Node 4 as the preferable candidate. However, this assessment would be incorrect. The actual performance of the cluster depends fundamentally on the majority consensus within the cluster. The leader-based consensus algorithm requires a majority of nodes to progress. This means that for any given cluster of n nodes, there is a need for the majority of $n/2 + 1$ nodes to be operational and able to communicate to make progress. More specifically majority can be expressed as:

$$M = \left\lfloor \frac{n}{2} \right\rfloor + 1, \tag{5}$$

where, brackets denote the floor function, which rounds down to the nearest whole number. This ensures that the majority is always more than half of the total number of nodes.

Hence, the determination of the most appropriate leader requires consideration of the majority principle within the cluster, which led to the definition of the new approach.

The new methodology defines a strategy for identifying the optimal leader in a network by considering the majority rule within a cluster. It emphasizes the selection of a node that utilizes the fastest communication channels within the cluster. The first step is to determine the smallest number of nodes required to form a majority that will govern decision-making. Subsequently, nodes are ordered by their communication speed, from the most to the least efficient. Within this prioritized group, the strategy involves calculating the average speed, determining the median, or identifying the single slowest link. Through this assessment, the node that is best positioned to take on the leadership role is selected, to expedite decision-making and enhance the overall efficiency of the cluster.

In mathematical terms, identifying the best leader within a cluster involves several steps that can be outlined as follows:

1. Calculate majority (M): The majority, M, is determined for the cluster. If the total number of nodes in the cluster is n, then M is calculated by formula 5.
2. Sort latencies: Let $L = \{l_1, l_2, ..., l_n\}$ represent the set of latencies from one node to all other nodes in the cluster. The latencies in L are then sorted in ascending order to obtain $L_{sorted} = \{l_{(1)}, l_{(2)}, ..., l_{(n)}\}$, where $l_{(1)} \le l_{(2)} \le ... \le l_{(n)}$
3. Select first M latencies: From the sorted set L_{sorted}, the first M latencies are selected, forming a subset $L_M = \{l_{(1)}, l_{(2)}, ..., l_{(M)}\}$.
4. Calculate metric to define the best leader: Use average, median or identify the slowest link within the quorum (majority).

Let us describe these metrics mathematically. The average is given by:

$$\bar{l} = \frac{1}{M} \sum_{i=1}^{M} l_{(i)} . \tag{6}$$

The median is defined as the middle value of L_M when M is odd, or the average of the two middle values when M is even:

$$M = l_{\left(\frac{M+1}{2}\right)} if(M\_is\_odd)$$

or

$$M = \frac{l_{\left(\frac{M}{2}\right)} + l_{\left(\frac{M+1}{2}\right)}}{2} if(M\_is\_even). \tag{7}$$

The slowest link within the majority is defined by:

$$\max(L_M) = \max\{l_1, l_2, ...l_M\}. \tag{8}$$

These values serve as a metric to evaluate the suitability of a node for leadership, based on its latency performance relative to the majority of nodes in the cluster.

When selecting an optimal metric for leader election, the leader's impact on the system overall throughput must be carefully considered. In environments where rapid decision-making is critical, such as in real-time systems where any delay in node communication could lead to significant consequences, focusing on the slowest link might be the most prudent approach. This metric is particularly telling as it encapsulates the performance threshold for the entire quorum; the system cannot progress faster than its slowest participating node. Therefore, the slowest link essentially sets the pace for the majority decision-making process. To ensure that the chosen metric aligns with the system demands and operational reality, it is imperative to base the decision on empirical evidence from real-world experiments, which provide a tangible measure of the system under typical operational loads and conditions.

While each metric offers potential benefits for leader election, the priority for immediate and efficient decision-making in critical environments guides the selection towards the slowest link metric for the present. This parameter is crucial as it sets the operational tempo for the entire quorum, with the system speed being inherently linked to the pace of its slowest node, ensuring that decisions are reached only once the entire majority has completed its tasks.

In light of these considerations, the approach to leader selection in distributed systems is reimagined through a quorum-based algorithm that prioritizes the stability of the network slowest link, ensuring the elected leader is the one most capable of maintaining system integrity during periods of latency variation. The algorithm, detailed below (Figure 3), introduces a methodical process that not only evaluates the total latency from a prospective leader to other nodes but also considers the maximum latency affecting the majority of the cluster. By doing so, it accounts for the critical path of communication which is essential for consistent and reliable decision-making. The pseudo-code representation of this novel leader selection process encapsulates a method that is both empirical and adaptive, serving as a blueprint for building resilient distributed systems in the face of fluctuating network conditions.

The methodology for optimal leader selection is in place; however, the specific mechanisms within the cluster to effectively enact this selection remain undefined. It is necessary to detail the procedures that will smoothly incorporate leader selection into the existing fabric of the cluster's operational protocol.

Algorithm 1 Leader Selection Based on Slowest Link in Quorum

```
procedure FindBestLeaderBasedOnSlowestLink($nodes$)
    $n \leftarrow \text{length}(nodes)$
    $majority \leftarrow \lfloor n/2 \rfloor + 1$
    $bestLeader \leftarrow \text{null}$
    $bestWorstLatency \leftarrow \infty$
    for all $candidate \in nodes$ do
        $latencies \leftarrow$ GetLatenciesFromCandidateToOthers($candidate, node$
        $sortedLatencies \leftarrow$ SortLatenciesAscending($latencies$)
        $majorityLatencies \leftarrow sortedLatencies[1 \ldots majority]$
        $slowestLatencyInMajority \leftarrow$ FindMaxLatency($majorityLatencies$)
        if $slowestLatencyInMajority < bestWorstLatency$ then
            $bestWorstLatency \leftarrow slowestLatencyInMajority$
            $bestLeader \leftarrow candidate$
        end if
    end for
    return $bestLeader$
end procedure
function GetLatenciesFromCandidateToOthers($candidate$, $nodes$)
    // Retrieves all latencies from candidate to other nodes
    // Returns a list of latencies: [l1, l2, ..., ln]
end function
function SortLatenciesAscending($latencies$)
    // Sorts latencies in ascending order
    // Returns the sorted list of latencies
end function
function FindMaxLatency($latencies$)
    // Finds the maximum latency in the given list
    // Returns the maximum latency
end function
```

Figure 3. Quorum-based leader selection algorithm

A. SELECTION PROCESS

In typical leader-based consensus algorithms like Raft, leader reelection is initiated when a follower node fails to receive regular "heartbeat" messages from the current leader within a predetermined timeout period, indicating a potential leader failure. The follower then transitions to a candidate state, increments its term count, and seeks votes from other nodes in the cluster. To become the new leader, the candidate must secure a majority of the votes from the cluster. Once elected, it begins transmitting new heartbeats to assert control and ensure log consistency across the cluster. This mechanism allows the system to swiftly recover from leader failures, enhancing its resilience and ensuring uninterrupted operation. The predefined timeout for these heartbeat messages is set randomly within a specific range, such as 100 to 300 milliseconds, to ensure that at least one node will time out sooner than others, facilitating a smooth transition to a new leader and effectively giving one node a head start.

The problem arises because the selection of the new leader is effectively random, as the timeout for heartbeat messages is set randomly, inadvertently giving one node an advantage. This approach, which uses a timeout for leader selection, simplifies the Raft algorithm, making it straightforward to implement. Therefore, it is beneficial to retain the concept of timeouts. To refine this process, it would be more strategic to adjust the system so that the most suitable node has a shorter timeout, allowing it to take the lead more quickly than the others.

As mentioned earlier, this simple random process also often leads to frequent leader reelections due to network delays.

Let us focus on establishing a process to determine the correct timeout for nodes within the cluster. This will aid in choosing the right leader if an election is imminent and prevent unnecessary elections from starting.

Given that the algorithm is distributed and there is no leader to oversee the process of selecting a new leader (because the leader is deemed unreachable at the time of selection), it is crucial for each node to independently determine its timeout. The challenge is that each node must base its decision on its knowledge and ensure it will be elected if its capabilities are the most suitable for the cluster's needs.

Another challenge is that each node's knowledge is restricted and does not reflect the overall state of the cluster. Independently, a node can measure its response times to other nodes, an essential factor for selecting the best leader. Therefore, the initial step is to establish a process where all nodes assess their delays to all other nodes. This measurement is crucial for evaluating their suitability for the role of leader.

Let us define the first step more specifically, each node N_i periodically assesses the delay $D_k(i,j)$ relative to every other node N_j using a schedule determined by an administrator-defined interval Δt. This delay measurement is performed at times $t_k = k\Delta t$ for each consecutive interval $k = 1,2,3,\ldots$ The setup ensures that nodes consistently and systematically measure and update their inter-node delay metrics according to the periodicity set by Δt, facilitating efficient network monitoring and management. It is crucial that the administrator carefully selects the interval Δt based on the cluster's computational capabilities and the current state of the network. Since generating ping messages, although computationally inexpensive, consumes CPU resources and increases network traffic, Δt should be optimized to balance the frequency of measurements with the potential impact on system performance. An overly frequent assessment could unnecessarily burden the network, while infrequent measurements might not capture significant changes in node latency in a timely manner. Therefore, Δt must be judiciously chosen to ensure efficient network monitoring without compromising the overall functionality and responsiveness of the cluster. A beneficial aspect of this approach is that for relatively stable networks, the timeout Δt can be set to longer intervals, ensuring that evaluations are not triggered too frequently. Additionally, this process can be further refined by assigning a unique offset to each node's timing, preventing simultaneous evaluations across all nodes and thereby reducing the risk of overwhelming the network at any single point in time.

Having defined the evaluation process, let us outline the concept for determining timeout. As a reminder, our goal is for each node to calculate a timeout based on delays to other nodes. This timeout should be brief enough to give the node a head start in the election process if it is best suited for leadership within the cluster. Additionally, this process must occur without direct communication with other nodes to maintain the effectiveness and simplicity crucial for distributed systems.

At the same time, this timeout should be flexible enough to cover temporary fluctuations of network (meaning some link experience temporary problems) and at the same time, it should be fixed to some value to make sure it will not be waiting for the failed node.

To express the process of defining the timeout value in a consensus algorithm under variable network conditions, it is evident that while this timeout should incorporate the delay of the slowest link in the majority, symbolized as $\max(L_M)$, it should not merely be set to this value. This approach is inadequate because an accurately calculated timeout must also be capped at a certain maximum $T_{\max}$ to ensure the system reconfiguration in case of actual node failure, and not just due to transient delays. This cap is ideally set by an administrator who understands the specific utilization requirements and constraints of the system.

The formula for setting the adaptive timeout T can thus be expressed as follows:

$$T = \left(\frac{\max(L_M)}{100}\right) * T_{\max} . \tag{9}$$

To address the issue of potential simultaneous timeouts across multiple nodes in the cluster when the network delays are uniformly distributed, it is indeed crucial to introduce randomness to the timeout setting as suggested by the Raft consensus protocol. This randomness helps in reducing the likelihood of all nodes initiating leader elections simultaneously due to experiencing timeouts at the same moment. We can modify the formula to incorporate a random component while maintaining the fundamental principles of the original setup.

$$T = \left(\frac{\max(L_M)_i + rand(a,b)}{100}\right) * T_{\max} , \tag{10}$$

where $rand(a,b)$ generates a random number between a and b (a and b provided by administrator).

Let us identify the potential drawbacks of this approach.

Firstly, using a simple constant such as 100 definitely will not work here, especially if network delays increase substantially, causing the result of multiplication to increase considerably.

Now, consider the next scenario involving a 5-node cluster (refer to Figure 4, where other nodes are assumed but not displayed). It is assumed that Node 2 is better suited to be a leader, given its latency, the maximum latency within the quorum is less than that of node 3: $\max(L_M)_2 < \max(L_M)_3$. Considering network delays, there is a possibility that node 3 may reach its timeout before the message from node 2 arrives, prompting node 3 to initiate a new election. Although node 2 has a better potential to win the election, this scenario highlights the need for further improvement.

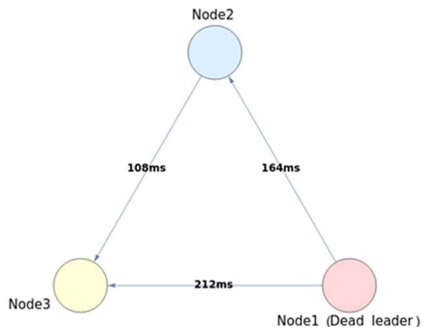


Figure 4. Cluster communication delay visualization: node-to-node latency mapping

Without further ado, let us present the solutions to those problems and refine the formula for the timeout.

Firstly, the process by which nodes transmit heartbeat messages needs modification. These messages should include information about delays experienced by the leader to each node and also convey the current maximum latency $\max(L_M)$ value. Additionally, each node receiving a heartbeat should measure the delay from the sending node to itself.

With this updated process, each node can gather comprehensive information about the current state of the cluster, enabling more informed decision-making in case of disruptions. It is acknowledged that during the early stages of execution, nodes may not yet receive this information. However, this is acceptable as the cluster's state is not at risk at this time; the information is primarily needed for process tuning and adjustment.

Initially, each node has access to the maximum latency values of the other nodes, $\max(L_M)$ represented as the set Θ:

$$\Theta = \{\max(L_M)_1, \max(L_M)_2, \ldots, \max(L_M)_{n-1}\} . \tag{11}$$

Each node can then independently assess whether it is the most suitable candidate for the role of leader using the condition:

$$\max(L_M)_i = \min(\Theta) . \tag{12}$$

Should a node determine that it is not the optimal choice for leadership, it is imperative that it calculates its waiting time so that delays do not negatively influence the election process. Thus, the node will compute its waiting time using the formula:

$$T_i = \max\left(\left(\frac{\max(L_M)_i}{\max(\Theta)}\right) \times T_{\max}, T_{\max} \times (1 - \partial_{me})\right) + rand(a,b) + \min\left(T_{dlbc} + T_{bccc} - T_{dlcc}, \delta_c\right), \tag{13}$$

where:

$\max(L_M)_i$ represents the delay of the slowest link within the majority of candidate i;

$rand(a,b)$ - a function to introduce randomness within a range (a and b provided by the administrator);

∂_{me} represents a value ranging from 0 to 1, which specifies the fraction of $T_{\max}$ to be used as the minimum value of T_i;

$T_{\max}$ is the maximum timeout value set by the administrator;

δ_c is the maximum timeout limit set to counteract communication delays within the cluster (specified by the administrator);

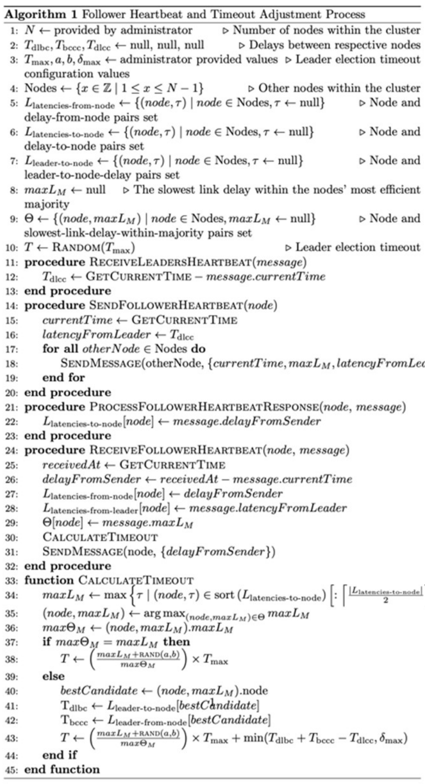
T_{dlbc} denotes the timeout from the assumed dead leader to the best candidate;

T_{bccc} represents the timeout from the best candidate to the current candidate;

T_{dlcc} stands for the timeout from the dead leader to the current candidate.

A node that presumes itself to be the optimal leader will not utilize the second part of the formula; therefore, the comprehensive formula that addresses both scenarios is as follows:

$$T_i = \begin{cases} \max\left(\left(\dfrac{\max(L_M)_i}{\max(\Theta)}\right)\times T_{\max}, T_{\max}\times\left(1-\partial_{me}\right)\right)+rand(a,b)+\min\left(T_{dlbc}+T_{bccc}-T_{dlcc},\delta_c\right) & if\ \max(L_M)_i=\max(\Theta) \\ \max\left(\left(\dfrac{\max(L_M)_i}{\max(\Theta)}\right)\times T_{\max}, T_{\max}\times\partial_{me}\right)+rand(a,b) & if\ \max(L_M)_i\neq\max(\Theta) \end{cases}. \quad (14)$$



```
Algorithm 1 Follower Heartbeat and Timeout Adjustment Process
N ← provided by administrator    ▷ Number of nodes within the cluster
$T_{dlbc}, T_{bccc}, T_{dlcc}$ ← null, null, null    ▷ Delays between respective nodes
$T_{max}, a, b, \delta_{max}$ ← administrator provided values    ▷ Leader election timeout configuration values
Nodes ← $\{x \in \mathbb{Z} \mid 1 \le x \le N-1\}$    ▷ Other nodes within the cluster
$L_{latencies\text{-}from\text{-}node} \leftarrow \{(node, \tau) \mid node \in Nodes, \tau \leftarrow null\}$    ▷ Node and delay-from-node pairs set
$L_{latencies\text{-}to\text{-}node} \leftarrow \{(node, \tau) \mid node \in Nodes, \tau \leftarrow null\}$    ▷ Node and delay-to-node pairs set
$L_{leader\text{-}to\text{-}node} \leftarrow \{(node, \tau) \mid node \in Nodes, \tau \leftarrow null\}$    ▷ Node and leader-to-node-delay pairs set
$maxL_M$ ← null    ▷ The slowest link delay within the nodes' most efficient majority
$\Theta \leftarrow \{(node, maxL_M) \mid node \in Nodes, maxL_M \leftarrow null\}$    ▷ Node and slowest-link-delay-within-majority pairs set
$T \leftarrow$ RANDOM($T_{max}$)    ▷ Leader election timeout
procedure RECEIVELEADERSHEARTBEAT(message)
    $T_{dlcc}$ ← GETCURRENTTIME − message.currentTime
end procedure
procedure SENDFOLLOWERHEARTBEAT(node)
    currentTime ← GETCURRENTTIME
    latencyFromLeader ← $T_{dlcc}$
    for all otherNode ∈ Nodes do
        SENDMESSAGE(otherNode, {currentTime, $maxL_M$, latencyFromLeader})
    end for
end procedure
procedure PROCESSFOLLOWERHEARTBEATRESPONSE(node, message)
    $L_{latencies\text{-}to\text{-}node}[node]$ ← message.delayFromSender
end procedure
procedure RECEIVEFOLLOWERHEARTBEAT(node, message)
    receivedAt ← GETCURRENTTIME
    delayFromSender ← receivedAt − message.currentTime
    $L_{latencies\text{-}from\text{-}node}[node]$ ← delayFromSender
    $L_{latencies\text{-}from\text{-}leader}[node]$ ← message.latencyFromLeader
    $\Theta[node] \leftarrow message.maxL_M$
    CALCULATETIMEOUT
    SENDMESSAGE(node, {delayFromSender})
end procedure
function CALCULATETIMEOUT
    $maxL_M \leftarrow \max\left\{\tau \mid (node,\tau) \in sort(L_{latencies\text{-}to\text{-}node})\left[: \left\lceil \frac{|L_{latencies\text{-}to\text{-}node}|}{2} \right\rceil\right]\right\}$
    $(node, maxL_M) \leftarrow \arg\max_{(node, maxL_M)\in\Theta} maxL_M$
    $max\Theta_M \leftarrow (node, maxL_M).maxL_M$
    if $max\Theta_M = maxL_M$ then
        $T \leftarrow \left(\frac{maxL_M + RAND(a,b)}{max\Theta_M}\right) \times T_{max}$
    else
        bestCandidate ← $(node, maxL_M)$.node
        $T_{dlbc} \leftarrow L_{leader\text{-}to\text{-}node}[bestCandidate]$
        $T_{bccc} \leftarrow L_{leader\text{-}from\text{-}node}[bestCandidate]$
        $T \leftarrow \left(\frac{maxL_M + RAND(a,b)}{max\Theta_M}\right) \times T_{max} + \min(T_{dlbc} + T_{bccc} - T_{dlcc}, \delta_{max})$
    end if
end function
```

Figure 5. Follower heartbeat and timeout configuration process

The node calculation process is outlined in pseudocode below (Figure 5). This procedure specifies that each node responds to a 'follower heartbeat', enabling the measurement of the delay from the node to itself. In certain scenarios, it may be feasible to simplify this process by assuming that the delay from the node is equivalent to the delay to the node, which holds true in most configurations.

III. EXPERIMENTAL RESULTS

Let us try to use the algorithm to calculate the leader election timeout for a cluster configuration [26]. Assume a cluster of five nodes (see Figure 6), node 1 is considered to be a leader with the following parameters defined:

$$T_{\max}=300,\ a=1,\ b=10,\ \delta_c=50,\ \partial_{me}=0.4$$

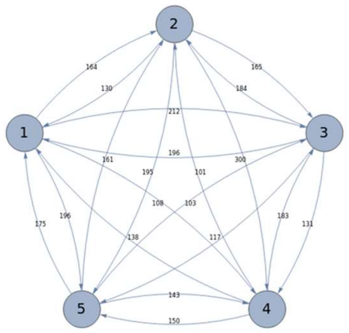


Figure 6. Five-node cluster

Network delays within the cluster are shown in Figure 7.

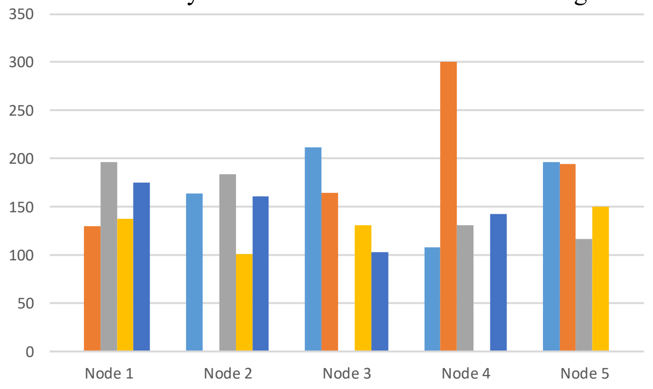


Figure 7. Network delays within the cluster

Table 2. Network delays within the cluster

-	Node 1	Node 2	Node 3	Node 4	Node 5
Node 1	-	164	212	108	196
Node 2	130	-	165	300	195
Node 3	196	184	-	131	117
Node 4	138	101	131	-	150
Node 5	175	161	103	143	-

Assuming the cluster has been operational for an extended period and all nodes are well-informed of the cluster state through the 'follower heartbeat' process, each node will use the defined algorithm to calculate the timeout if the leader (node 1) experiences a sudden failure.

Considering the issues defined earlier, the timeouts, as calculated by the designated algorithm, should be:

- adaptive to the current network delays within the cluster;
- optimally short for the most suited node to react quicker than others;

- sufficiently long to accommodate the current state of the network, yet short enough to initiate a timeout promptly if the leader is deemed unresponsive;
- staggered to prevent simultaneous timeouts among the nodes and avoid the split-brain scenario within the cluster.

Firstly, to ascertain which node is best equipped to assume leadership, we will employ formulas 7 and 8 to compute the $maxL_m$ values for each node. These calculations will help identify the node with the optimal characteristics for leadership based on current network conditions and node responsiveness, results of calculations are presented in Table 3.

Table 3. Cluster delay properties according to proposed algorithm

Node	$\{l_1, l_2, \dots l_{N-1}\}$	sort$(\{l_1, l_2, \dots l_M\})$	$max(L_M)$
1	$\{164, 212, 108, 196\}$	$\{108, 164, 196\}$	196
2	$\{130, 165, 300, 195\}$	$\{165, 195, 300\}$	300
3	$\{196, 184, 131, 117\}$	$\{117, 131, 184\}$	184
4	$\{138, 101, 131, 150\}$	$\{101, 131, 150\}$	150
5	$\{171, 161, 103, 143\}$	$\{103, 143, 161\}$	161

As it can be observed, node 4 has the minimum value of $\max(L_M) = 150$ and should be considered the best candidate to assume leadership of the cluster. Therefore, the algorithm should yield a value that complies with the before mentioned assumptions.

Let us use formula 14 to calculate the timeouts for each node in the cluster, the results and main intermediate computation results are shown in Table 4.

Table 4. Final and main intermediate computation results

Node	$max(\Theta)$	T_{dlbc}	T_{bccc}	T_{dlcc}	$Rand(a,b)$	T
1	300	108	138	0	7	253
2			101	164	8	353
3			131	212	3	214
4			0	108	4	154
5			150	196	1	212

As observed, node 4 has a timeout value of 154 milliseconds. When compared to the delay between the current leader (node 1) and the best candidate (node 4), which is 108 milliseconds, the timeout value perfectly aligns with the requirements. It is adaptive, closely approximating 108 milliseconds, and is optimally short, allowing to react more quickly than others and trigger reelection swiftly in case of abnormalities in network delays. Additionally, it is sufficiently long to accommodate the current state of the network, which helps prevent unnecessary reelections due to fluctuations in the network. Overall, the timeouts are staggered to prevent simultaneous timeouts among the nodes, thus enhancing system stability.

Although the experiment yields strong results, additional testing and modeling are essential to further validate the new approach. Rigorous experimentation will ensure that the algorithm performs as expected under various network conditions and scenarios, ultimately confirming its reliability and efficiency in real-world applications.

VI. CONCLUSIONS

This article has explored a sophisticated strategy for enhancing consensus algorithms in distributed systems, particularly in the face of unstable network conditions. The innovative approach introduced involves dynamically adjusting the consensus parameters to better align with the real-time state of network connectivity, thereby maintaining stable system performance under varying network conditions.

The research presented provides a solid theoretical foundation for understanding how adaptive mechanisms can be integrated into existing consensus frameworks to mitigate the challenges posed by network instability. This includes the significant enhancement of system resilience, ensuring that distributed systems remain functional and consistent despite network disruptions.

Future research should focus on the empirical validation of the proposed modifications, examining their effectiveness across different network scenarios and configurations. This will require rigorous testing and simulations to confirm the anticipated improvements in system reliability and performance. The continuation of this work will not only validate the theoretical models presented but also contribute to the ongoing refinement of consensus algorithms.

In summary, the article contributes to the broader discourse on distributed system design by proposing methods that adjust to network variability, thereby enhancing the robustness and reliability of these systems. This approach represents a significant step forward in distributed computing, ensuring that systems are better equipped to handle the complexities and unpredictabilities of real-world network conditions.

References

[1] S. Zhuravel, M. Klymash, O. Shpur, and O. Lavriv, "Achieving consistency and consensus of distributed infocommunication systems," *Proceedings of the 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, Lviv, Ukraine, February 22-26, 2022, pp. 386-389. https://doi.org/10.1109/TCSET55632.2022.9767019.

[2] S. Zhuravel, O. Shpur, and Y. Pyrih, "Method of achieving consensus in distributed service," *Infocommunication Technologies and Electronic Engineering,* vol. 2, no. 2, pp. 58–66, 2022. https://doi.org/10.23939/ictee2022.02.058.

[3] G. Stafford, *LAN network stability: measure response time of a wireless vs. ethernet-based LAN,* 2021, [Online]. Available at: https://www.kaggle.com/code/garystafford/network-stability-notebook/input.

[4] W. Zhong, C. Yang, W. Liang, J. Cai, L. Chen, J. Liao and N. Xiong, "Byzantine fault-tolerant consensus algorithms: A survey," *Electronics*, vol. 12, no. 18, 3801, 2024. https://doi.org/10.3390/electronics12183801.

[5] R. Hao, X. Dai, X. Xie, "Doppel: A BFT consensus algorithm for cyber-physical systems with low latency," *Journal of Systems Architecture*, vol. 148, 103087, 2024. https://doi.org/10.1016/j.sysarc.2024.103087.

[6] Z. Hussein, M. Salama, and S. El-Rahman, "Evolution of blockchain consensus algorithms: a review on the latest milestones of blockchain consensus algorithms," *Cybersecurity*, vol. 6, no. 30, 2023. https://doi.org/10.1186/s42400-023-00163-y.

[7] K. Venkatesan and S. Rahayu, "Blockchain security enhancement: an approach towards hybrid consensus algorithms and machine learning techniques," *Scientific Reports*, vol. 14, p. 1149, 2024. https://doi.org/10.1038/s41598-024-51578-7.

[8] F. Nawab, M. Sadoghi "Consensus in data management: From distributed commit to blockchain," *Foundations and Trends in Databases*, vol. 12, issue 4, pp. 221-364, 2023. http://doi.org/10.1561/1900000075.

[9] Y. Xiao, N. Zhang, W. Lou, Y. Hou, "A survey of distributed consensus protocols for blockchain networks," *IEEE Commun. Surv. Tutorials*, vol. 22, issue 2, pp. 1432–1465, 2020. https://doi.org/10.1109/COMST.2020.2969706.

[10] S. Fahim, S. M. Katibur Rahman, S. Mahmood, "Blockchain: A comparative study of consensus algorithms PoW, PoS, PoA, PoV," *International Journal of Mathematical Sciences and Computing (IJMSC)*, vol. 9, no. 3, pp. 46-57, 2023. https://doi.org/10.5815/ijmsc.2023.03.04.

[11] Y. Li, Y. Fan, L. Zhang, and J. Crowcroft, "RAFT consensus reliability in wireless networks: probabilistic analysis," *IEEE Internet of Things Journal*, vol. 10, issue 14, pp. 12839-12853, 2023. https://doi.org/10.1109/JIOT.2023.3257402.

[12] H. Knudsen, J. Notland, P. Haro, T. Ræder, and J. Li, "Consensus in blockchain systems with low network throughput: a systematic mapping study," *Proceedings of the 3rd Blockchain and Internet of Things Conference,* July 2021, pp. 15-23. https://doi.org/10.1145/3475992.3475995.

[13] M. Kleppmann, *Designing Data-Intensive Applications*, O'Reilly UK Ltd, 2017, 614 p.

[14] F. Palacios, E. Quesada, H. La, S. Salazar, S. Commuri, and L. Garcia Carrillo, "Adaptive consensus algorithms for real-time operation of multi-agent systems affected by switching network events," *International Journal of Robust and Nonlinear Control*, vol. 27, issue 9, 2016. https://doi.org/10.1002/rnc.3687.

[15] N. Lutsiv, T. Maksymyuk, M. Beshley, O. Lavriv, V. Andrushchak, et al., "Deep semisupervised learning-based network anomaly detection in heterogeneous information systems," *Computers, Materials & Continua*, vol. 70, issue 1, pp. 413-431, 2022. https://doi.org/10.32604/cmc.2022.018773.

[16] B. Wang, S. Liu, H. Dong, X. Wang, W. Xu, J. Zhang, P. Zhong, Y. Zhang, "Bandle: asynchronous state machine replication made efficient," *Proceedings of the Nineteenth European Conference on Computer Systems*, Association for Computing Machinery, April 2024, pp. 265–280. https://doi.org/10.1145/3627703.3650091.

[17] A. Guru, H. Mohapatra, B. Mohanta, C. Altrjman, A. Yadav, "A survey on consensus protocols and attacks on blockchain technology," *Applied Sciences*, vol. 13, issue 4, 2604, 2023, https://doi.org/10.3390/app13042604.

[18] Y. Sang, H. Shen, Y. Tan, N. Xiong, "Efficient protocols for privacy preserving matching against distributed datasets," In: Ning, P., Qing, S., Li, N. (eds) *Information and Communications Security. ICICS 2006. Lecture Notes in Computer Science*, vol 4307. Springer, Berlin. Heidelberg. https://doi.org/10.1007/11935308_15.

[19] N. El Rharbi, H. Atteriuas, A. Younes, A. Harchaoui, O. Izem "A comparative study of the recent blockchain consensus algorithms," *Proceedings of the E-Learning and Smart Engineering Systems (ELSES 2023).* Atlantis Press, 2023, pp. 316-327. https://doi.org/10.2991/978-94-6463-360-3_32.

[20] S. Liu, R. Zhang, C. Liu, et al., "An improved PBFT consensus algorithm based on grouping and credit grading," *Sci Rep 13*, 13030, 2023. https://doi.org/10.1038/s41598-023-28856-x.

[21] N. Hagshenas, M. Mojarad, H. Arfaeinia, "A fuzzy approach to fault tolerant in cloud using the checkpoint migration technique," *International Journal of Intelligent Systems and Applications (IJISA)*, vol. 14, no. 3, pp. 18-26, 2022. https://doi.org/10.5815/ijisa.2022.03.02.

[22] S. Jamuna, P. Dinesha, K. Shashikala, K. Kishore Kumar, "Design and implementation of reliable encryption algorithms through soft error mitigation," *International Journal of Computer Network and Information Security (IJCNIS)*, vol. 12, no. 4, pp. 41-50, 2020. https://doi.org/10.5815/ijcnis.2020.04.04.

[23] N. Razali, I. Isa, S. Sulaiman, N. Noor, M. Osman, "CNN-Wavelet scattering textural feature fusion for classifying breast tissue in mammograms," *Biomedical Signal Processing and Control*, vol. 83, pp. 104683, 2023. https://doi.org/10.1016/j.bspc.2023.104683.

[24] A. Yazdinejad, R. Parizi, A. Dehghantanha, K. Choo, "P4-to-blockchain: A secure blockchain-enabled packet parserfor software defined networking," *Comput. Secur.*, vol. 88, p. 101629, 2020. https://doi.org/10.1016/j.cose.2019.101629.

[25] J. Yusoff, Z. Mohamad, M. Anuar, "A review: consensus algorithms on blockchain," *Journal of Computer and Communications*, vol. 10, issue 09, pp. 37–50, 2022. https://doi.org/10.4236/jcc.2022.109003.

[26] N. Peleh, S. Zhuravel, O. Shpur, O. Rybytska, "Structured and unstructured log analysis as a methods to detect DDoS attacks in SDN networks," *Internet of Things (IoT) and Engineering Applications*, pp. 1-9, Sept. 2021. https://doi.org/10.1007/978-3-030-92435-5_12.

S. ZHURAVEL is currently pursuing a PhD at Lviv Polytechnic National University in Lviv, Ukraine, a program he began in 2021. Alongside his studies, he serves as an assistant lecturer in the Department of "Telecommunications" at the same university. He also has over five years of experience as a Software Engineer, specializing in the development of enterprise applications. His research interests include distributed systems, consensus algorithms, and distributed computing.

O. SHPUR, PhD, as Associate Professor of the Department of "Telecommunications" at Lviv Polytechnic National University. Her research interests include principles of building and functioning of data center in distributed service systems, adaptation of data center operations to the integration of multi-task CloudNFV/Big Data structures, cloud-technology, SDN.

M. KLYMASH is now the Chief of Telecommunication Department, Lviv Polytechnic National University, Ukraine. He received his PhD in optical data transmission, location and processing systems from Bonch-Bruevich Saint-Petersburg State University of Telecommunications, Saint Petersburg, Russia, in 1994 Honored member of Ukrainian Communications Academy. The topics of his current interest of research include distributed networks, cloud computing, convergent mobile networks, big data, software defined networks and 5G heterogeneous networks.

•••