

EDRaft: An Energy-Driven Consensus Protocol for Energy-Harvesting IoT Networks

Glauber Barros   [Universidade Federal do Rio Grande do Norte | glauber.mendes.112@ufrn.edu.br]
Gibeon Aquino  [Universidade Federal do Rio Grande do Norte | gibeon@dimap.ufrn.br]

 Department of Mathematics and Applied Informatics, Campus Universitário, Lagoa Nova, Natal, RN, 59078-970, Brazil.

Received: 08 April 2026 • Accepted: 23 September 2026 • Published: 27 October 2026

Abstract In resource-constrained IoT networks, limited battery capacity and intermittent energy sources necessitate consensus protocols capable of efficiently adapting to energy scarcity. However, standard consensus algorithms such as Raft are energy-agnostic, operating under the implicit assumption that all nodes have stable power supplies, risking the election of Leaders with critical energy levels and triggering frequent re-election cycles that accelerate network-wide depletion. To address this, we propose EDRaft, a phase-based, energy-driven evolution of Raft in which nodes continuously adjust their behavior, and the cluster's required majority, in response to real-time energy availability. We evaluate EDRaft against three baseline protocols, Raft, E-Raft, and Nakagawa's protocol, using a discrete-event simulator, across nine scenarios varying cluster size and client workload, under both non-harvesting and harvested energy conditions. EDRaft's Leader consistently outlasts its counterpart under the baseline protocols, by 9.8 to 40.8% under active workload and up to 78.7% when idle, translating into 44 to 115% higher request throughput at the network level.

Keywords: Energy-Harvesting, Consensus Algorithms, Raft Algorithm, Energy-Driven, Internet of Things

1 Introduction

Distributed consensus algorithms are the backbone of reliable systems, ensuring fault tolerance and data consistency across varied network topologies [Shetty *et al.*, 2019]. Among these, the Raft algorithm [Ongaro and Ousterhout, 2014] has emerged as a standard due to its understandability and strong consistency guarantees, being widely adopted in cloud infrastructure. However, the expansion of distributed systems into the Internet of Things (IoT) and Edge Computing imposes strict energy constraints.

Energy harvesting, which converts natural energy sources into usable electrical power, is gaining recognition as a promising solution for self-sustaining and environmentally responsible deployments [Lucia *et al.*, 2017]. Nevertheless, this approach presents its own set of challenges, including the instability and unpredictability of harvested energy levels [Sliper *et al.*, 2019]. To mitigate these fluctuations, the energy-driven computing [Sliper *et al.*, 2019; Merrett and Al-Hashimi, 2017; Balsamo *et al.*, 2015; Tuncel *et al.*, 2022; Fletcher *et al.*, 2017; Jayakumar *et al.*, 2014] paradigm has emerged as a highly effective countermeasure. By dynamically adapting a system's behavior and operational states to real-time energy availability, this approach prioritizes survivability and resilience over maximum performance during periods of scarcity.

The standard Raft algorithm is energy-agnostic, implicitly assuming that all nodes operate with stable and continuous power supplies. This assumption is critical in energy-harvesting environments, where power availability is inherently variable and unpredictable. Since Raft treats all respon-

sive nodes as equally capable, it may unintentionally elect a Candidate that, despite being active, has critical battery levels or is subject to environmental fluctuations. Such a Leader lacks the energy stability to sustain the heavy workload of log replication, leading to abrupt failures and frequent re-elections that degrade system availability and accelerate network-wide energy depletion.

Several studies have proposed modifications to the standard Raft protocol to address energy constraints through distinct architectural adaptations. For instance, the work in [Nakagawa and Hayashibara, 2018] focuses on reducing message overhead by introducing a suspended Follower role, where a subset of nodes is actively deactivated and only resumed when the cluster lacks the necessary quorum. Alternatively, the Pirogue algorithm [Pâris and Long, 2015] proposes a more dynamic approach, implementing quorum adjustments based on the number of active nodes and introducing witness nodes (participants that store only metadata) to accommodate devices with limited computational resources. Finally, E-Raft [Li *et al.*, 2021] integrates an energy model directly into the leader election process, restricting candidacy to nodes that exceed a specific energy threshold and assigning low-energy nodes to a passive *alsoran* state to prevent premature depletion.

While recent literature proposes energy-efficient adaptations for Raft and other consensus mechanisms, most works focus on static energy conservation strategies or energy thresholds for the leader election process, neglecting the unpredictable nature of energy harvesting profiles. In contrast, this paper introduces Energy-Driven Raft (EDRaft), an energy-aware approach in which IoT end devices (nodes) may adopt

non-optimal strategies to reduce energy consumption during periods of energy scarcity in order to survive in unstable and energy-scarce scenarios caused by the energy harvesting process. This approach increases the system's fault tolerance and, consequently, its operational availability in these scenarios. The main contributions of this work are:

- **EDRaft: An energy-aware distributed consensus algorithm.** Nodes running EDRaft dynamically adjust their behavior based on real-time energy availability, effectively handling the instability of harvested energy levels.
- **A reproducible energy harvesting model.** We model solar harvesting as a deterministic day/night cycle modulated by two layers of variability, a daily radiation factor and a Markov-chain cloud state, both fitted from real solar radiation and cloud cover observations.

This paper extends previous work [da Silva Barros *et al.*, 2023], which introduced behavior changes based on energy availability exclusively for the Leader role, overlooking the importance of Follower nodes in maintaining cluster operation. The major new additions introduced relative to this prior work are: (i) the introduction of Leader behavior adjustments during energy scarcity, in which the Leader suspends non-essential Follower nodes once it enters the Attention phase, retaining only the minimum number required to satisfy the majority (quorum), and defers client request replication to be batched with the next scheduled heartbeat once it enters the Alert phase; (ii) the introduction of a dynamic quorum mechanism, in which the majority required for consensus is recalculated against the number of currently live nodes as Followers deplete their energy, subject to a floor of two active nodes; (iii) the proposal of behavior adjustments for Follower nodes during energy scarcity, implemented through a Suspension Mode in which Followers only respond to service requests from a Leader or Candidate when no majority state exists; (iv) an expanded experimental study that includes all node roles and analyzes how energy consumption is affected when a Follower assumes the Proxy role; and (v) a broader range of execution scenarios, including executions with an energy harvesting model.

The remainder of this paper is structured as follows. Section 2 presents the main related works relevant to our solution. Section 3 describes the architecture of the proposed algorithm. Section 4 details the experimental setup and execution scenarios. Section 5 presents the results and discussions. Finally, Section 6 concludes the paper and outlines future directions for this research.

2 Related Works

Given the scarcity of studies in the literature that jointly address consensus algorithms and energy-driven computing, this work considers related proposals that introduce variations aimed at improving energy efficiency or performance in consensus protocols such as Raft, Paxos [Lamport, 2001], and Byzantine fault-tolerant algorithms. For each selected work, we present a brief description of the core proposal, its primary contributions, and the results reported by the authors.

The solution proposed in [Nakagawa and Hayashibara, 2018] introduces an extension to the Raft algorithm aimed at improving energy efficiency through the dynamic management of node activity. The approach incorporates a new suspended role for Followers. At the start of execution, the Leader actively suspends a subset of Follower nodes based on a predefined parameter that specifies the percentage of nodes to deactivate. A Follower node designated as suspended is ineligible for leader election processes and does not receive messages from the active Leader. This reduction in active participants decreases the number of messages required to reach consensus within the cluster, thereby reducing energy consumption. A suspended node can return to an active role upon receiving a ResumeMessage from either a Leader or a Candidate node. This message is typically issued when multiple node failures occur and the cluster lacks the quorum needed to proceed with consensus or election.

In [Pâris and Long, 2015], the authors introduce an algorithm called Pirogue, which offers a lighter and more dynamic alternative to Raft. The algorithm's core innovation lies in its dynamic quorum adjustment, which adapts the number of required votes for consensus based on the current number of active nodes in the cluster. For example, in a cluster with four nodes, the quorum initially equals three. If one node fails, the quorum reduces to two and continues decreasing until a tie occurs; at this point, the system resolves the tie based on the voting policy adopted. The authors also propose a novel node state, termed a witness, which can replace one or two nodes in the cluster. Witness nodes store only metadata about the disseminated data, allowing them to participate in consensus decisions without holding the actual payload. This approach enables the integration of low-computation devices into the consensus process. Across all configurations tested, Pirogue delivered availability levels comparable to those achieved by a five-node Raft cluster.

The E-Raft consensus mechanism, proposed in [Li *et al.*, 2021], integrates the original Raft algorithm with an energy model. While the log replication process remains unchanged from Raft, the leader election procedure undergoes specific modifications. In this mechanism, a Follower whose election timer expires and who has not received communication from the Leader will only transition to the Candidate state if its energy level exceeds a predefined threshold. Otherwise, the node assumes a new role introduced in E-Raft, called *alsoran*. When a node enters the *alsoran* state, it lowers its energy threshold and must resend information to update other nodes. This mechanism aims to balance the excessive energy consumption typically associated with the Leader role by introducing constraints that prevent premature energy depletion.

In [Bhat *et al.*, 2023], the authors introduce the Energy Efficient BFT-SMR (EESMR) as a consensus mechanism designed to address both Byzantine fault tolerance and energy efficiency. EESMR operates through two core procedures, namely steady-state operation and view-change operation. The steady-state operation functions when the Leader behaves correctly, while the view-change operation is triggered upon the detection of Byzantine faults, requiring a change in leadership. During steady-state execution, the Leader proposes blocks to the replicas in each round. After receiving a

proposal, each replica broadcasts the block to the other replicas and updates its local state with the content proposed by the Leader.

In [Fernandez Blanco *et al.*, 2023], the authors propose a consensus mechanism named PyCloudIoT Consensus Model (PCM), which aims to enhance consensus efficiency by reducing the number of network messages. This optimization is achieved through the formation of subgroups within the cluster. The proposed mechanism operates on two hierarchical levels, the gateway level and the cluster level. At the gateway level, the system elects a strong Leader, referred to as the Resource Manager (RM), whose decisions override those of other nodes. The RM is responsible for managing the infrastructure topology, organizing nodes into subgroups, and handling new client requests. Upon receiving a request, the RM assigns it to a designated subgroup, ensures its replication within that subgroup, and subsequently returns the final processing result to the client. Consensus is also implemented at the cluster level. Within each subgroup, the RM elects a weak Leader, termed the Coordinator Orchestrator (CO), whose decisions can be overridden by the RM. Each CO is responsible for determining whether consensus has been reached within its subgroup through a local voting process among Follower nodes.

To provide a comprehensive overview of the state-of-the-art, Table 1 classifies the discussed works according to their underlying consensus protocol, fault model, core optimization strategy, target network environment, and impact on message overhead. As shown, existing solutions primarily focus on either modifying leader election criteria [Li *et al.*, 2021] or dynamically adjusting node participation [Nakagawa and Hayashibara, 2018; Pâris and Long, 2015; Fernandez Blanco *et al.*, 2023] to conserve energy. While works such as [Bhat *et al.*, 2023] address Byzantine environments, they do not prioritize message reduction as a primary energy-saving mechanism. Work [Li *et al.*, 2021] is classified as only partially energy-driven, since although its energy threshold prevents a depleted node from assuming the Leader role, thereby indirectly avoiding that role's higher cost, it does not modify the node's ongoing behavior within its current role, unlike a continuous, reversible adaptation to energy state. The remaining works, [Nakagawa and Hayashibara, 2018; Pâris and Long, 2015; Bhat *et al.*, 2023; Fernandez Blanco *et al.*, 2023], are classified as not energy-driven. [Nakagawa and Hayashibara, 2018; Pâris and Long, 2015; Fernandez Blanco *et al.*, 2023] reduce energy consumption as a consequence of their design, but none of their decision logic takes a node's actual energy level as an input, while [Bhat *et al.*, 2023] reduces energy cost entirely through fixed, without any node adapting its behavior to its own energy condition during execution. Our proposed solution differentiates itself by jointly applying energy-driven adaptation to both the Leader's transmission behavior and the Followers' participation.

3 Proposed Algorithm

The modifications introduced by EDRaft transform the standard Raft algorithm into an energy-aware protocol tailored for scenarios relying on energy harvesting. This adaptation

addresses the instability of harvested energy levels by dynamically adjusting node operations within the cluster.

Our approach categorizes energy scarcity into distinct phases, each representing a specific level of criticality based on the node's available energy (AE). For each phase, the algorithm defines role-specific behavioral adjustments designed to maximize the overall operational lifespan of the cluster under energy-constrained conditions.

While EDRaft retains Raft's core processes (leader election and log replication), it enhances the election mechanism to prioritize nodes with higher energy availability. This strategy promotes leadership stability and mitigates the energy costs associated with frequent re-elections.

3.1 Leader Election

When a Leader fails, the cluster initiates an election process to restore operations. This procedure involves extensive message exchanges, which inherently increases energy consumption across the cluster. EDRaft optimizes this process by biasing the election towards nodes with higher energy reserves.

To implement this strategy, EDRaft embeds energy meta-data into the *RequestVote* messages that Candidates broadcast to Followers. This modification enables Followers to assess a Candidate's energy status before granting a vote. Upon receiving a request, a Follower compares the Candidate's energy level against its own. If the Candidate possesses less energy than the Follower, the vote is denied. This policy ensures that leadership is assigned to the most capable node, thereby promoting system longevity and stability.

3.2 Energy Phases

To support the proposed approach, we identified distinct energy phases that an energy-harvesting device experiences throughout its operational lifecycle. Energy availability fluctuates over time. A node may harvest abundant energy and operate without restrictions, or it may face prolonged scarcity requiring significant behavioral adaptation. In extreme cases, a node may approach an operational shutdown due to energy depletion. By clearly defining these phases, we establish strategies that allow nodes to rationalize energy usage and sustain operations for as long as possible.

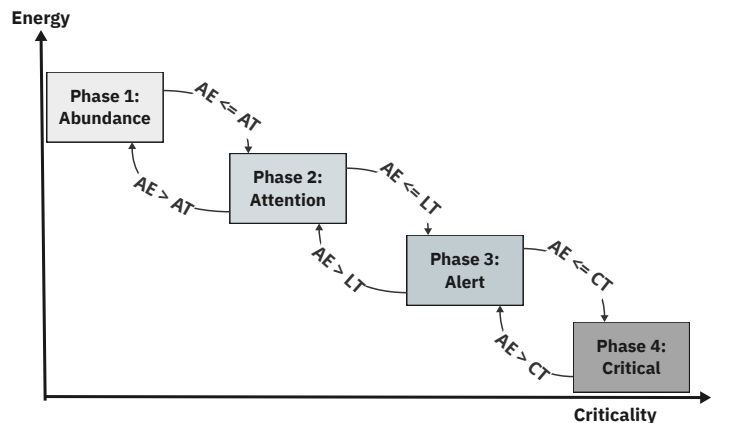


Figure 1. Transition logic between phases.

Table 1. Comparative analysis of related works addressing consensus and energy-driven computing.

Work	Algorithm Basis	Fault Model	Core Focus	Network Type	Reduces Messages	Energy-Driven
[Nakagawa and Hayashibara, 2018]	Raft	Non-Byzantine	Node Activity	Wired	Yes	No
[P�aris and Long, 2015]	Raft-based	Non-Byzantine	Dynamic Quorum	Wired	Yes	No
[Li <i>et al.</i> , 2021]	Raft	Non-Byzantine	Leader Election	Wireless	No	Partially
[Bhat <i>et al.</i> , 2023]	BFT-SMR	Byzantine	Steady-state	Wired	No	No
[Fernandez Blanco <i>et al.</i> , 2023]	Custom	Non-Byzantine	Hierarchical	Wireless	Yes	No
This Work	Raft	Non-Byzantine	Leader Election/Node Activity	Wireless	Yes	Yes

EDRaft defines four phases: **Abundance**, **Attention**, **Alert**, and **Critical**. Figure 1 illustrates the transition logic and the conditions triggering each state. We define three thresholds that delineate these phases, the **Attention Threshold** (AT), the **Alert Threshold** (LT), and the **Critical Threshold** (CT). For instance, if the available energy (AE) of a device in the Abundance phase falls below the AT, the node transitions to the Attention phase. These thresholds depend on factors such as the harvesting system’s capacity, battery size, and the device’s power consumption profile. Since these parameters vary by hardware and application, threshold values must be calibrated for each specific deployment.

The rationale behind this four-phase design (specifically the distinction between *Attention* and *Alert*) is to separate preventive measures from restrictive ones. A three-phase model would force the system to either trigger resource-intensive mechanisms (such as the Proxy role) prematurely or delay energy-saving actions until the node is nearly depleted. The *Attention* phase applies low-cost adjustments to prevent the need for the drastic behavior changes that characterize the *Alert* phase.

3.2.1 Abundance Phase

The Abundance phase represents a condition where the device maintains sufficient stored energy relative to its operational requirements. In this state, the node operates without restrictions, fully utilizing its computational resources to execute all planned tasks.

3.2.2 Attention Phase

This phase represents a cautionary condition where the device retains considerable energy but has dropped below the Abundance level. While there is no immediate risk of failure, this phase signals the need for proactive measures. Consequently, nodes in this phase undergo minimal adaptations aimed at reducing energy consumption while maintaining full system functionality.

3.2.3 Alert Phase

In this phase, the device enters a state of significant energy scarcity. The operational principle shifts to survival, nodes accept substantial limitations on their activities compared to the previous phases. The objective is to drastically reduce consumption to endure the period of scarcity while sustaining only essential capabilities.

3.2.4 Critical Phase

The Critical phase represents the highest level of urgency. Upon entering this state, the node initiates a cluster shut-

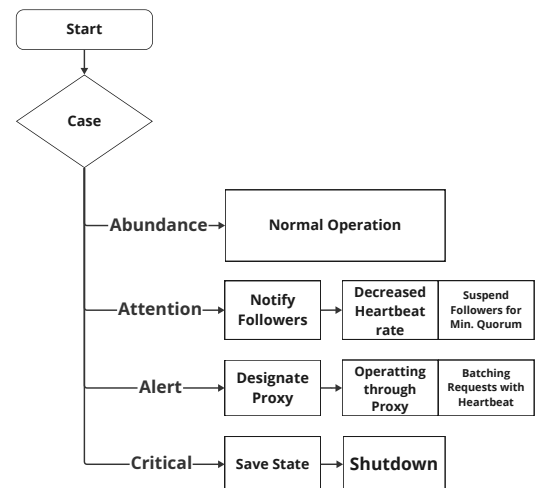
down process to preserve system integrity and prevent data corruption. To ensure this process completes successfully, the Critical Threshold (CT) must be set higher than the total energy required to execute the reliable shutdown sequence.

3.3 Behavior Adaptation

During Raft execution, a node assumes one of three roles defined by the algorithm: Leader, Follower, or Candidate. The Leader is primarily responsible for managing cluster operations, handling external client requests, and replicating data to other nodes. The Follower plays a passive role, maintaining its local data replica and monitoring the presence of an active Leader. A node transitions to the Candidate role upon detecting the absence of a Leader, initiating a new election process. Since the Candidate role is transient and strictly necessary for electing a new Leader after a failure, EDRaft does not introduce specific behavioral adaptations for this role. However, given the distinct responsibilities and varying energy consumption levels of the other roles, EDRaft defines specific behavioral adjustments for both the Leader and the Follower.

3.3.1 Leader

In the standard Raft protocol, the Leader performs the most intensive workload, resulting in the highest energy consumption among all roles. Consequently, EDRaft implements the most substantial operational adjustments for the Leader during energy-critical phases. Figure 2 presents a simplified flowchart illustrating the Leader’s behavior across these phases. In the **Abundance phase**, the Leader operates without any modifications intended to limit energy consumption or computational resources.

**Figure 2.** Leader’s operation flowchart.

During the **Attention phase**, the Leader increases the interval between heartbeat messages sent to the Followers. This adjustment reduces the message exchange frequency, thereby decreasing the Leader's energy expenditure. To support this extended interval, it is crucial that Followers adjust their election timeouts to prevent premature elections caused by the delayed heartbeats. To synchronize this change, the Leader broadcasts a dedicated Remote Procedure Call (RPC) named *HeartbeatIncrementRPC*. Upon receiving this RPC, a Follower updates its election timeout according to the increment value provided by the Leader. When the Leader's available energy recovers to a level corresponding to the Abundance phase, the heartbeat interval extension introduced upon entering the Attention phase is reversed. This reversal reuses the same *HeartbeatIncrementRPC*, now carrying a dedicated reset flag instead of a positive increment value. Upon receiving this message, a Follower restores its heartbeat expectation and corresponding election timeout to their original.

In addition to this heartbeat interval increment, upon entering the Attention phase, the Leader begins reassessing, at every periodic check, how many active Followers are strictly necessary to satisfy the cluster's quorum requirement. Any Follower beyond this minimum is placed into the Suspension Mode, a concept adapted from Nakagawa and Hayashibara [2018], through an explicit instruction from the Leader. The Leader sends a *SuspendFollowerRPC* to the targeted Follower, ensuring it does not misinterpret the resulting silence as Leader failure and spuriously trigger an election. Should the number of active Followers later fall below what is required for quorum, whether due to further energy depletion or the departure of an active node, the Leader reactivates a suspended Follower to restore the necessary quorum size. The Follower's own participation in this mechanism is described in the following subsection.

To conserve energy during the **Alert phase**, the Leader delegates log replication duties to a Proxy node, selected from the Followers with the highest available energy (AE). To identify the most suitable Candidate, the Leader must maintain up-to-date energy information for the cluster. To this end, an additional argument named *lifeTime* was incorporated into the Follower's response to the *AppendEntriesRPC*, conveying the node's current energy level. Thus, every response from a Follower includes its energy status. Once a Proxy is designated, the Leader and the Proxy verify log consistency. If inconsistencies are detected, synchronization rounds are executed. After successfully activating the Proxy, the Leader reduces its communication overhead by sending new client entries and heartbeats solely to the Proxy, rather than broadcasting to the entire cluster. This approach significantly lowers the Leader's transmission costs. The Proxy assumes the responsibility of replicating received entries to the remaining Followers, applying the same majority-based consensus strategy as the Leader. Upon reaching consensus, the Proxy reports the result to the Leader, which in turn forwards the response to the client. Upon the Leader's return to a higher energy phase, the Proxy receives a *ShutdownProxyRPC* from the Leader, after which it resigns as Proxy and reverts to the Follower role.

In parallel with Proxy delegation, from the Alert phase onward, the Leader no longer replicates each client request

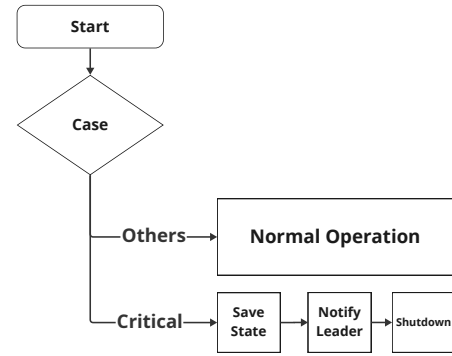


Figure 3. Proxy's operation flowchart.

immediately upon arrival; instead, incoming requests are held locally and replicated only once bundled together with the next scheduled heartbeat, reducing the number of individual transmissions required to serve a given volume of client traffic. To prevent this batching mechanism from indefinitely delaying requests under sustained load, a maximum bundle size triggers an early replication once reached, independently of the regular heartbeat schedule.

In the **Critical phase**, the Leader ceases accepting new client requests. It immediately replicates any latest log entries not yet propagated to the Proxy, including any request still awaiting batched transmission, ensuring their distribution to the rest of the cluster. This strategy prevents the Leader from failing before disseminating its most recent state, thereby avoiding potential log inconsistencies in a subsequently elected Leader. Once all pending entries are replicated, the Leader performs a checkpointing procedure to persist its state and initiates a shutdown process. After the Leader stops sending heartbeats, the cluster detects the failure and enters a new election process, in which the Followers elect a new Leader.

3.3.2 Proxy

EDraft introduces a new role, the Proxy, whose primary responsibility is to assist the Leader during periods of energy scarcity, thereby mitigating the risk of leadership failure and subsequent cluster unavailability. Once the Leader designates a Proxy, it delegates the communication burden to this node; all heartbeat signals and client log entries are forwarded to the Proxy, which then assumes the responsibility for disseminating this information to the remaining Followers.

To facilitate this, the Proxy utilizes a dedicated Remote Procedure Call named *AppendEntriesProxyRPC*. This mechanism allows Followers to distinguish that the log entries originate from an intermediary, ensuring the correct interpretation and processing of the received data.

During its active period, the Proxy assumes a pivotal role in maintaining cluster functionality. Since it mediates the entire log replication process, applying energy-saving restrictions to the Proxy's operations could compromise the cluster's overall availability. To address this, operational adjustments to the Proxy are applied exclusively during the **Critical phase**. In all preceding phases (Abundance, Attention, and Alert), the Proxy continues to perform its full set of responsibilities, as illustrated in Figure 3.

Upon entering the **Critical phase**, the Proxy executes a

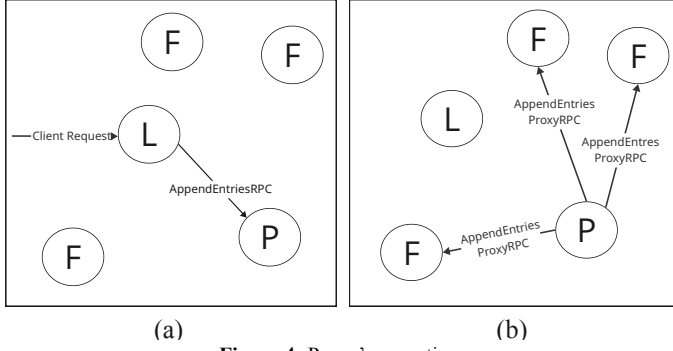


Figure 4. Proxy's operation.

checkpointing procedure to persist its state and immediately notifies the Leader of its inability to continue. This notification is sent via a dedicated RPC named *ShutDownProxyRPC*. After dispatching this message, the Proxy relinquishes its role, reverts to a standard Follower state, and proceeds with its shutdown sequence. Upon receiving the *ShutDownProxyRPC*, the Leader initiates the selection of a new Proxy to maintain cluster stability.

The data replication flow involving the Proxy occurs in two distinct steps. First, the Leader (L) receives a client request and forwards it to the active Proxy (P), as illustrated in Figure 4(a). Second, the Proxy replicates the received entries to the Follower nodes (F), effectively acting as the cluster's broadcaster, as shown in Figure 4(b).

3.3.3 Follower

The Follower role entails a reduced set of responsibilities compared to the Leader. While the Leader manages client requests and log replication, Followers are primarily responsible for maintaining passive consistency, receiving log entries and initiating elections only upon detecting a Leader failure.

In standard Raft, the failure of a single Follower is typically tolerable. However, in small-scale clusters or energy-constrained environments, multiple Follower failures can jeopardize the quorum, halting the system. Consequently, EDRaft introduces specific adjustments to Follower behavior during the **Alert** and **Critical** phases, designed to preserve cluster stability. Figure 5 presents a simplified flowchart outlining these modifications.

During the **Alert phase**, nodes adopt the Suspension Mode. Prior to entering this mode, the Follower notifies the Leader

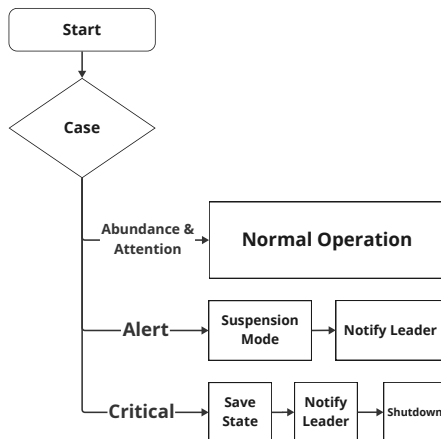


Figure 5. Follower's operation flowchart.

via a *SuspendedFollowerRPC*. Upon acknowledging this message, the Leader adds the node to a suspended list and ceases sending regular heartbeats or new log entries to it. These suspended nodes are only contacted if the number of active nodes becomes insufficient to reach a consensus quorum. If a suspended Follower recharges sufficiently to reach the Abundance phase, it exits suspension and notifies the Leader, which then reintegrates the node into the active replication set.

The two Suspension Mode triggers, whether initiated by the Follower itself or by the Leader, operate on the same underlying Suspension Mode and compose without conflict. A Follower already suspended for one reason is simply recognized as already suspended for the other, without any duplicated effect. Two interactions between them are, however, worth noting. First, the Leader's resumption logic is not conditioned on why a Follower became suspended; when quorum falls short, the Leader reactivates a suspended Follower regardless of that Follower's own remaining energy, prioritizing cluster availability over the individual node's conservation strategy, which may in turn accelerate that Follower's own depletion once reactivated. Second, a Follower may recover sufficient energy to exit suspension on its own initiative even while the Leader continues to regard it as surplus with respect to quorum; in this case, the Leader reassesses the active Follower count at the next periodic check and, if the recovered Follower remains unnecessary for quorum, suspends it again.

In the **Critical phase**, the Follower triggers a graceful shutdown. It first ceases all operations and notifies the Leader via a *ShutdownFollowerRPC*. Subsequently, it performs a checkpointing procedure to persist its state. Crucially, while in this phase, the Follower remains strictly disconnected; it cannot process requests or participate in elections, even if the cluster halts due to a lack of quorum. This strict isolation prioritizes hardware integrity over service availability.

Once the node recovers sufficient energy, it rejoins the cluster as a Follower, regardless of its previous role (e.g., Leader or Proxy). To rejoin, the node broadcasts a *ReintegrateFollowerRPC* to all peers. This broadcast is necessary because the recovering node may be unaware of the current Leader's identity, especially if leadership changed during its downtime. Although sent to the entire cluster, only the current active Leader processes this request and synchronizes the returning node.

3.4 Dynamic Quorum

Building on the dynamic quorum concept introduced by Pirogue [Pâris and Long, 2015], EDRaft adopts its own dynamic quorum mechanism, centered on a single alive-node counter that determines the majority required for both replication and election at any given moment.

The Leader maintains this counter directly. Whenever a Follower depletes its energy, it notifies the current Leader of this event, and the Leader decrements the counter accordingly. Conversely, when a previously depleted node reintegrates (Section 3.3.3), the Leader increments the counter upon synchronizing its log. This shrinking applies specifically to energy depletions, which are events the cluster becomes explicitly aware of through these notifications. Other categories of failure, such as network partitions or crashes unrelated to

energy exhaustion, are not accounted for by this mechanism and do not reduce the required majority. To make this value available to the rest of the cluster, the counter is embedded as an additional field in the *AppendEntriesRPC* the Leader already sends to Followers on every heartbeat and log replication. A future Candidate therefore uses the most recent counter value received from the previous Leader as the basis for the majority it must obtain to win.

This shared counter is what allows the mechanism to preserve Raft's original safety and consistency guarantees. Because a node only resumes counting toward this value once the Leader has synchronized its log upon reintegration, no node can participate in a majority calculation with an outdated log, preserving the consistency of any majority-based decision as the number of active nodes changes.

A minimum of two active nodes is always required, regardless of how small the cluster's alive count becomes, preventing a lone surviving node from unilaterally certifying its own leadership or committing entries by itself.

4 Experimental Study

In this experimental study, we conduct a comparative analysis of EDRaft, Raft, E-Raft, and Nakagawa's protocol under equivalent operational settings across diverse scenarios, assessing quality attributes including performance, availability, and energy efficiency.

4.1 Methodology

This work assumes that a faulty node is one whose energy has reached zero; other types of failure, such as network partitions or crashes unrelated to energy exhaustion, are not considered in this experimental study. Accordingly, the simulated network is assumed to be fully connected and partition-free throughout each execution, with reliable message delivery subject only to the propagation delay and message size costs; the sole failure mode modeled is node unavailability resulting from energy depletion. Evaluating EDRaft under these additional failure modes is left as future work, described in Section 6.

Four protocols, Raft [Ongaro and Ousterhout, 2014], EDRaft, E-Raft [Li *et al.*, 2021], and Nakagawa's protocol [Nakagawa and Hayashibara, 2018], were evaluated using a custom discrete-event simulator implemented in MATLAB¹. The simulator models each node as an independent entity with its own role (Leader, Follower, Candidate, and, where applicable, Proxy or Witness), position in a two-dimensional deployment field, and available energy. All inter-node communication, including heartbeats, vote requests, log replication, and protocol-specific control messages, is represented as discrete events carrying an explicit propagation delay and message size, processed in simulated-time order by an event queue.

Each simulation run executes all four protocols under identical conditions of node placement, client request timing, and random seed, isolating protocol behavior from incidental variation in the execution environment. To mitigate the effect

of this variation on the reported results, each scenario was executed ten times, with a new random seed independently drawn for each execution, while the same seed remained shared across all four protocols within a given execution as described above. The results reported in Section 5 correspond to the average across these ten executions.

The experimental evaluation is organized into two complementary sets of executions. The first, referred to as *role-isolated evaluation*, assigns an effectively unlimited energy budget to every role except the one under study, so that the energy trajectory of a single role, Leader, Follower, or Proxy, can be observed without interference from confounding effects such as re-elections triggered by the depletion of other nodes or quorum renegotiation. This isolation allows the energy impact of each individual behavioral adjustment to be measured directly, attributable specifically to the mechanism under evaluation, avoiding secondary cluster dynamics. Harvesting is disabled throughout this evaluation set, since this would introduce variability unrelated to the mechanism under study and obscure the attribution that role isolation is designed to provide.

The second set, referred to as *network-level evaluation*, assigns finite energy to every node simultaneously, allowing all roles to interact as they would during actual operation, including Leader failures, re-elections, and quorum renegotiation triggered by the depletion of any node. This set captures network-wide metrics, such as total available energy and request throughput, that are properties of the cluster as a whole, and that cannot be meaningfully observed under the role-isolated configuration.

4.1.1 Energy Model

Each node maintains a local variable representing its currently available battery capacity, in Joules. This variable is decremented according to a radio energy model adapted from Heinzelman's work on LEACH protocol [Heinzelman *et al.*, 2000], which expresses transmission, reception, and idle-listening costs as closed-form functions of message size and inter-node distance. Transmission cost is computed as shown in Eq. (1), where k is the message size in bits and d is the Euclidean distance to the receiver. The exponent α and amplifier coefficient ϵ switch between a free-space (d^2) and a multipath (d^4) regime at a crossover distance determined by the ratio of the two amplifier coefficients.

$$E_{tx}(k, d) = E_{elec} \cdot k + \epsilon \cdot k \cdot d^\alpha \quad (1)$$

Reception cost is defined by Eq. (2), which depends only on message size. Every message transmission or reception immediately and synchronously decrements the sender's and receiver's local energy variables at the simulated instant it occurs. Each node continuously compares its local energy variable, expressed as a percentage of its initial capacity, against the predefined Attention, Alert, and Critical thresholds, and autonomously classifies its current phase without any centralized coordination.

$$E_{rx}(k) = E_{elec} \cdot k \quad (2)$$

¹<https://www.mathworks.com/discovery/network-simulation.html>

Table 2. Open-Meteo Historical Weather API query parameters.

Parameter	Value
Location (lat, lon)	5.80°S, 35.34°W
Elevation	27 m
Start date	2024-01-01
End date	2024-03-30
Timezone	America/Fortaleza (UTC−3)
Weather model	ERA5
Hourly variables	Shortwave Radiation, Cloud Cover Total

4.1.2 Harvesting Model

The harvesting rate is modeled as a deterministic half-sine day/night cycle, extended with two additional layers of variability whose parameters were fitted from a real solar radiation and cloud cover dataset for Greater Natal, Rio Grande do Norte, Brazil (5.80°S, 35.34°W, 27 m elevation), between January 1 and March 30, 2024, obtained through the Open-Meteo Historical Weather API². Table 2 summarizes the exact query parameters used to obtain this dataset.

The harvesting rate at time t is given by:

$$r(t) = P_{peak} \cdot f_{day} \cdot m_{c(t)} \cdot d(t) \quad (3)$$

where P_{peak} denotes the maximum achievable harvesting rate, attained only when the remaining three factors are simultaneously at their maximum. The term $d(t) = \max(0, \sin(2\pi(t \bmod T_{day})/T_{day}))$ represents the underlying day/night cycle; it increases from zero at dawn to a maximum of one at solar noon, decreases back to zero at dusk, and remains at zero throughout the night, with T_{day} set to 86,400 s, a full 24-hour cycle. P_{peak} is set to 730 μ W. The two remaining terms, f_{day} and $m_{c(t)}$, introduce variability around this deterministic cycle and are the components fitted from the real dataset described above. Table 3 summarizes the calibrated values for both layers of variability, detailed below.

The factor f_{day} represents day-to-day variability in solar radiation. At the start of each simulated day, its value is drawn from $\mathcal{U}(f_{min}, f_{max})$ and held fixed until the following day, with f_{min} and f_{max} fitted to 0.39 and 1.0, respectively, set to the minimum and maximum daily radiation ratios observed in the dataset, each day's total radiation expressed as a ratio to the strongest day observed in the same window.

The factor $m_{c(t)}$ represents intra-day variability caused by cloud cover, and is determined by a cloud state $c(t)$ that evolves through a discrete-time Markov chain, so that a given state persists over time, consistent with the temporal persistence of cloud cover observed in the dataset. The three cloud states correspond to clear, partly cloudy, and overcast conditions, classified from the dataset's cloud cover fraction; each state's multiplier was fitted to 0.905, 0.838, and 0.754, respectively, set to the average clearness index observed within it, the ratio between the radiation observed in a given hour and the highest radiation observed at that same hour-of-day across the full ninety-day window. A transition out of the current state occurs with probability modeled in Equation 4,

$$P_{transition} = 1 - e^{-\Delta t / \tau_{c(t)}} \quad (4)$$

where Δt is the simulation tick interval and $\tau_{c(t)}$ is the mean dwell time of the current cloud state, fitted to 13,464 s, 5,976 s,

Table 3. Calibrated harvesting model parameters.

Cloud state	Cover threshold	Multiplier (m_c)	Dwell time (τ_c)
Clear	< 33%	0.905	13,464 s (3.74 h)
Partly cloudy	33–66%	0.838	5,976 s (1.66 h)
Overcast	> 66%	0.754	24,444 s (6.79 h)

Daily radiation factor: $f_{day} \sim \mathcal{U}(0.39, 1.0)$, redrawn once per simulated day.

and 24,444 s for the clear, partly cloudy, and overcast states, respectively, likewise obtained from the dataset, as the mean duration of consecutive same-category hours observed within it. At each simulation tick, this probability is evaluated against an independently drawn uniform random number in $[0, 1)$; a transition occurs if the drawn number falls below $P_{transition}$, and the state persists otherwise. The markedly longer persistence of overcast conditions relative to clear ones reflects a genuine asymmetry present in the data. When a transition occurs, the new state is drawn uniformly among the three cloud states considered.

The parameters f_{min} , f_{max} , m_c , and τ_c are fixed values obtained from the dataset, whereas the specific sequence of daily factors and cloud state transitions occurring during a given simulation run is generated stochastically at execution time. This allows simulations of arbitrary duration to be produced without replaying the original observations directly.

4.2 Setup

To evaluate the behavior of EDRaft against Raft, E-Raft, and Nakagawa's protocol under varying conditions, we defined a set of experimental parameters. Table 4 details the values assigned to each parameter.

The deployment field was set to $100m \times 100m$, a scale large enough for inter-node distance to meaningfully affect transmission energy cost under the physical radio model, while remaining consistent with the spatial scale typically considered for a single-cluster wireless sensor network deployment.

A critical parameter in this evaluation is the cluster size, as it directly influences the volume of messages transmitted by the Leader or Proxy and the number of vote requests exchanged during elections. We evaluated clusters with 5, 11, and 21 nodes, capable of tolerating up to 2, 5, and 10 simultaneous failures, respectively.

Another parameter is client throughput. We considered three variations, an idle cluster where nodes exchange only internal consensus messages, a low active cluster, where external clients submit 2 requests per second and a high active

Table 4. Configuration Parameters.

Parameters	Quantity
Field	100m x 100m
Nodes	5N, 11N, 21N
Client Requests	0req/s, 2req/s, 5req/s
Phases Thresholds	AT : 60% LT : 30% CT : 0.1%
Heartbeat Increment	0.2s
Initial Energy (EH OFF)	0.5J
Initial Energy (EH ON)	2J
Exec. Duration (EH OFF)	Until network depletion
Exec. Duration (EH ON)	172800s (2 days)
Max. EH Rate (P_{peak})	730 μ W

²<https://open-meteo.com/en/docs/historical-weather-api>

cluster, where external clients submit 5 requests per second. The inclusion of idle scenarios allows us to isolate the baseline energy cost of the consensus algorithm from the application workload. For the active scenarios, we selected a rate of 2 and 5 requests per second based on preliminary stress tests, identifying this as the maximum sustainable throughput without inducing timeouts under the defined configuration. The workload consisted exclusively of SET commands, where a client submits a new value for a specific key to be updated in the cluster. We restricted the workload to write requests (SET commands) because the Raft log replication process is triggered solely by state changes. Unlike read requests, write requests mandate the propagation of log entries to all Followers, ensuring that every request engages the full consensus mechanism and its associated energy costs.

The Attention threshold was set at 60%, intentionally above the halfway point of the node's capacity, to trigger the first, mildest adjustments early enough that heartbeat interval extension and quorum-based Follower suspension have sufficient remaining runway to meaningfully slow the node's depletion rate before more restrictive measures become necessary. The Alert threshold was set at 30% to preserve enough of the node's remaining capacity for its more drastic measures, namely Proxy delegation and client request batching, to have a meaningful impact on survival time; setting this threshold much lower would leave too little energy for these mechanisms to meaningfully extend the node's operation before the Critical threshold is reached. Finally, the Critical threshold was set at 0.1% to guarantee that the energy remaining at this point comfortably exceeds the cost of the node's final actions. Under the energy model's radio parameters and the deployment field's maximum inter-node distance, persisting local state and issuing the *ShutdownProxyRPC* together cost approximately $114 \mu\text{J}$ in the worst case; the energy available at the Critical threshold exceeds this cost by a factor of approximately 4.4 to 17.5, depending on the initial energy configuration, ensuring these operations reliably complete before the node is fully depleted.

Additionally, the increment for the heartbeat interval during the Attention phase was fixed at 0.2s. By varying these parameters, we defined nine execution scenarios (Scenarios 1–9). Table 5 presents the specific parameters for each scenario.

The initial energy differs between the harvesting disabled and harvesting enabled scenarios. Under harvesting disabled conditions, a lower initial energy of $0.5J$ was chosen to facilitate observation of the impact of the energy-saving behaviors adopted by EDRaft's roles. Under harvesting enabled conditions, a higher initial energy of $2J$ was required since energy replenishment through solar harvesting only becomes significant over a time scale of hours.

The execution duration is defined differently across the two harvesting conditions. Under harvesting disabled conditions, no fixed time limit is imposed; each run instead proceeds until the cluster becomes inoperative. Under harvesting enabled

conditions, by contrast, energy replenishment means a node may in principle never fully deplete, so a fixed execution duration of $172800s$ (2 days) was instead adopted. This duration was chosen to be long enough to encompass multiple full day/night cycles and several transitions between the cloud states described in the harvesting model.

4.3 Evaluated Metrics

To validate the proposed algorithm and assess the experimental results, we defined a set of evaluation metrics. Through this validation, we aim to measure the impact of operational changes on the energy consumption of the nodes, including the effects of a Follower assuming the Proxy role, the impact of a Follower entering Suspension Mode, and the overall impacts on the variation of each evaluated metric. The defined metrics are:

- i. **Operational Duration:** The total time a node remains active. This metric is crucial for comparing the algorithms' ability to prolong the Leader's tenure. A longer-lasting Leader reduces the frequency of leadership transitions and the associated energy overhead of new elections, which impacts the entire cluster.
- ii. **Total Processed Requests:** The total number of client requests successfully committed and replicated across the cluster. This metric assesses system availability, ensuring that requests are handled even during energy-critical periods. Since a fixed request rate is applied in active scenarios, the number of processed requests is directly proportional to the node's execution time, i.e., a longer operation results in a higher total throughput.
- iii. **End-to-End Latency:** The elapsed time between a client request arriving at the Leader and its response being sent to the client. This metric captures the cost of the energy-saving mechanisms that delay message propagation, namely request batching and Proxy relay, allowing the trade-off between energy savings and responsiveness to be assessed directly.
- iv. **Energy Consumption:** The instantaneous energy level of each node, sampled at 1-second intervals throughout the execution. This monitoring allows us to correlate energy consumption patterns with specific phase transitions and the operational adjustments triggered in each phase.

Unless noted otherwise, each of these metrics is reported in the section that follows as the mean value computed across the ten independent executions per scenario and protocol described in Section 4.2

5 Results and Discussion

This section presents the results obtained from the two evaluation sets defined in Section 4, followed by a discussion of the implications of each finding.

Table 5. Parameters For Each Scenario

Scenario	1	2	3	4	5	6	7	8	9
Nodes	5	5	5	11	11	11	21	21	21
Req/s	0	2	5	0	2	5	0	2	5

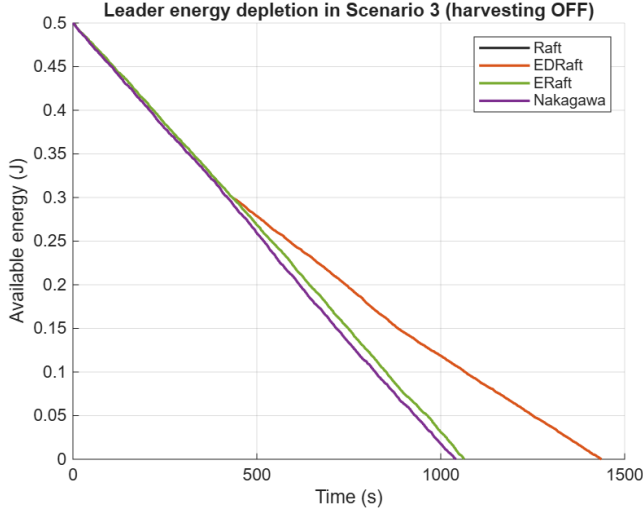


Figure 6. Leader energy depletion in Scenario 3 (harvesting OFF)

5.1 Role-Isolated Evaluation

5.1.1 Leader Energy Depletion

Across all nine scenarios evaluated, the EDRaft Leader survives longer than the corresponding Leader under the three baseline protocols (Raft, E-Raft, and Nakagawa), without exception. Under active client workload, the relative improvement in Leader duration over the Raft baseline ranges from a minimum of 9.8%, observed in Scenario 9, to a maximum of 40.8%, observed in Scenario 2, indicating that the advantage conferred by EDRaft's adaptive mechanisms is not confined to a particular configuration but persists across the full range of tested conditions. In the idle scenarios (Scenarios 1, 4, and 7), the relative improvement is considerably larger, reaching 78.7% in Scenario 1 and remaining above 12% even in Scenario 7, which corresponds to the largest cluster size evaluated under idle conditions, consistent with periodic heartbeat transmission constituting a proportionally larger share of the Leader's total energy expenditure in the absence of client-driven replication traffic.

Figure 6 presents the Leader's energy trajectory in Scenario 3, with harvesting disabled, for all four evaluated protocols, providing direct visual evidence of the mechanism underlying this improvement. For approximately the first 460 seconds of simulated time, all four curves remain coincident, declining at the same rate from the initial energy of 0.5 J to approximately 0.30 J. At this point, the EDRaft Leader completes its transition from the Abundance phase to the Attention phase, defined by the threshold $AT = 0.60$ applied to the initial energy, which corresponds to exactly 0.30 J in this scenario. From this point onward, the EDRaft curve visibly diverges from the three baseline protocols, declining at a markedly slower rate, a direct consequence of the two adjustments that take effect upon entering the Attention phase, the extension of the heartbeat interval, which reduces the frequency of radio transmissions performed by the Leader, and the suspension of non-essential Followers beyond the minimum required for quorum, which reduces the number of recipients the Leader must reach on each remaining transmission.

The EDRaft curve shows a further reduction in its rate of decline at approximately 870 seconds, when the available energy reaches approximately 0.15 J. This point corresponds

to the transition from the Attention phase to the Alert phase, defined by the threshold $LT = 0.30$ applied to the initial energy, which corresponds to exactly 0.15 J in this scenario. Upon entering the Alert phase, the Leader delegates broadcast responsibility to a designated Proxy node, replacing a full broadcast transmission with a single unicast transmission to the Proxy. In parallel, incoming client requests are no longer replicated immediately upon arrival, but held and batched together with the next scheduled heartbeat, further reducing the number of individual transmissions required to serve a given volume of client traffic. This further reduction in per-transmission energy cost is reflected in the additional flattening of the EDRaft curve beyond this point.

The three baseline protocols show no corresponding reduction in their rate of decline at either threshold, since none of them modifies its transmission behavior in response to its own remaining energy. Raft and E-Raft remain fully coincident throughout the simulation, both depleting at approximately 1062.9 seconds, while Nakagawa depletes slightly earlier, at approximately 1041.1 seconds. The EDRaft Leader, by contrast, remains operational until approximately 1437 seconds, corresponding to the duration reported for Scenario 3.

Table 6 presents the Leader survival duration, expressed in seconds, together with the total number of client requests successfully handled by the Leader before depletion, for each of the four evaluated protocols across all nine scenarios defined in Table 5. A methodological consequence of the role-isolated design is worth making explicit before discussing Table 6's values. Since this evaluation set assigns unlimited energy to every role except the one under study, the isolated Leader never undergoes a re-election, as no other node in the cluster can fail. E-Raft's only departure from standard Raft restricts candidacy during the election process; because no election occurs under this configuration, E-Raft's Leader depletes at exactly the same rate as Raft's Leader in every scenario, which Table 6 confirms numerically.

Nakagawa's protocol, in turn, modifies only Follower activity, not the Leader's own behavior, so its Leader duration coincides exactly with Raft's in the three idle scenarios (1, 4, and 7), where no client-driven traffic differentiates the two. Under active workload, Nakagawa's Leader diverges slightly from Raft's, in either direction depending on the scenario. Because a Follower placed in suspension no longer receives messages from the active Leader, the Leader's own transmission cost is reduced by an amount that depends on the distance to the specific Follower suspended, which varies with the random node placement of each scenario. This equivalence reflects that the role-isolated methodology by construction cannot expose mechanisms that act on roles other than the one under observation. The independent contributions of E-Raft and Nakagawa's protocol become visible once elections and Follower dynamics interact with the Leader, which is precisely what the network-level evaluation of Section 5.2 captures.

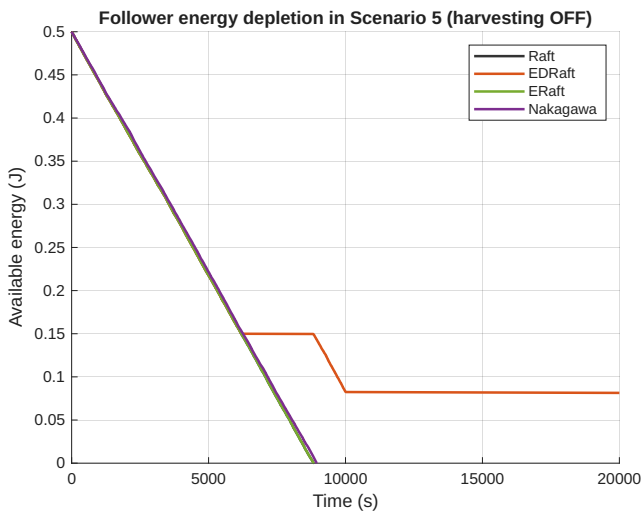
It is also observed that the Leader's total number of requests handled is consistently higher for EDRaft than for the baseline protocols in every loaded scenario. Since a longer operational duration allows the Leader to remain available to clients for a correspondingly longer period, this outcome follows directly from the extended survival time already discussed.

Table 6. Leader Duration Time (s)

Scenario	Raft	EDRaft	E-Raft ^a	Nakagawa ^b
1	21109.0	37720.2	21109.0	21109.0
2	2487.0	3502.6	2487.0	2483.6
3	1062.9	1437.2	1062.9	1041.1
4	16915.0	23383.4	16915.0	16915.0
5	2140.5	2692.4	2140.5	2136.0
6	925.1	1170.0	925.1	901.3
7	13323.0	14915.4	13323.0	13323.0
8	1760.9	2023.4	1760.9	1818.3
9	794.0	872.1	794.0	776.3

^a Identical to Raft in every scenario by construction of the role-isolated methodology; see the paragraph introducing this table.

^b Identical to Raft in the idle scenarios for the same reason; the small deviations under active workload are explained in the paragraph introducing this table.

**Figure 7.** Follower energy depletion in Scenario 5 (harvesting OFF).

5.1.2 Suspension Mode Evaluation

Survival duration does not constitute an appropriate metric for comparing Follower behavior across protocols. Under EDRaft's Suspension Mode, a suspended Follower ceases communication activity, and its energy consumption becomes negligible for an extended period, rendering its depletion time not directly comparable to that of the baseline protocols. The effect of Suspension Mode is therefore analyzed through the shape of the resulting energy trajectory.

Figure 7 illustrates this behavior. The EDRaft Follower depletes at the same rate as the baseline protocols, until entering Suspension Mode, at which point its energy trajectory becomes flat while the baselines continue depleting to full exhaustion. The EDRaft Follower is later reactivated, a behavior consistent with EDRaft's quorum-driven resumption mechanism, after which it resumes depleting and stabilizes at a lower energy level. In this scenario, the EDRaft Follower retains substantially more energy than the baseline protocols once the latter have been fully depleted.

Since a cluster typically contains several Followers simultaneously, this preservation effect applies to multiple nodes at once instead of to a single role, as is the case for the Leader, and consequently contributes disproportionately to the total network energy and survival results reported in the network-level evaluation that follows.

5.1.3 Proxy Energy Depletion

To evaluate the energy cost associated with assuming the Proxy role, the isolated Proxy scenario compares the EDRaft node designated as Proxy against a Raft Follower under identical conditions. In this configuration, the Leader is held within the Alert phase throughout the simulation, ensuring that a Proxy is assigned and remains active for the full duration of the run.

Across all defined scenarios, the EDRaft Proxy depletes substantially faster than the Raft Follower baseline. This finding indicates that the relay responsibility carries a real and consistent energy cost, one that is not offset, at the level of the individual node bearing it, by any of EDRaft's other energy-saving mechanisms. Table 7 reports the survival duration of both roles across all nine scenarios, together with the ratio of Proxy duration to Follower duration.

A more detailed examination of this ratio indicates that the magnitude of the Proxy's energy disadvantage is determined primarily by the client workload instead of by the size of the cluster. Under idle conditions (Scenarios 1, 4, and 7), the Proxy survives for approximately 24 to 28% of the Follower's duration, regardless of whether the cluster contains 5, 11, or 21 nodes. Under a workload of 5 requests per second (Scenarios 3, 6, and 9), this figure rises to approximately 46 to 53%, again with little variation across cluster size. Scenarios with an intermediate workload of 2 requests per second fall between these two extremes, ranging from approximately 43 to 49% across the three cluster sizes evaluated. This progression indicates that the Proxy's relative disadvantage narrows steadily as workload increases.

This increase in ratio with workload follows from the structure of the relay mechanism itself. The Proxy performs an additional transmission on every heartbeat, independent of whether that heartbeat carries a client request; consequently, this additional cost is largely fixed per unit of time. Under idle conditions, this fixed cost constitutes a substantial proportion of the Proxy's total consumption, producing the low ratios observed in Scenarios 1, 4, and 7. Under active workload, the cost of processing and replicating requests becomes considerable for both roles, diluting the relative weight of the Proxy's fixed cost and accounting for the progressively higher ratios observed under moderate and heavy workloads (Table 7). The ratios observed across different cluster sizes further indicate that this cost originates from the act of transmission itself, since the energy model attributes a single transmission cost calibrated to the most distant recipient.

Table 7. Proxy Duration Time (s)

Scenario	Follower (s)	Proxy (s)	Ratio
1	49067	13513	27.5%
2	8908	4310	48.4%
3	3950	2109	53.4%
4	49120	11801	24.0%
5	8857	3811	43.0%
6	4012	1850	46.1%
7	49049	13531	27.6%
8	8850	4305	48.6%
9	3959	2112	53.3%

The Proxy's energy cost is the counterpart to the Leader and Follower savings reported elsewhere in this evaluation. EDRaft's Proxy delegation mechanism does not eliminate the energy cost of broadcasting to the cluster; instead, it redistributes that cost from the Leader to whichever node currently holds the Proxy role. This redistribution is advantageous to the cluster as a whole whenever the Leader is the more energy-constrained of the two roles, since delaying the Leader's own depletion in exchange for a faster-depleting Proxy extends the period during which the cluster retains a functioning coordinator, avoiding the disruption and re-election cost of a Leader failure. This mechanism contributes, alongside EDRaft's other role-specific adjustments, to the higher request throughput and total network energy reported for EDRaft in the network-level evaluation that follows. From the perspective of the individual node designated as Proxy, however, the arrangement is energetically disadvantageous, and a node assigned to this role depletes its energy correspondingly faster than it would under any of the baseline protocols evaluated.

5.2 Network-Level Evaluation

5.2.1 Request Throughput

Across the scenarios evaluated, EDRaft consistently processes more client requests than the three baseline protocols, both with and without harvesting enabled, a direct consequence of the Leader and Follower energy savings already established in the role-isolated evaluation. Table 8 reports the number of client requests processed by each protocol across scenarios involving active client workload, under both conditions. A processed request is one whose entry reached a majority of the then-alive cluster. With harvesting disabled, the rela-

tive improvement over Raft ranges from 44% in Scenario 2 to 115% in Scenario 9. With harvesting enabled, the same pattern holds across all six scenarios, with the relative improvement over Raft ranging from 14.3% in Scenario 3 to 39.7% in Scenario 9.

This improvement follows directly from the energy savings documented for the Leader and Follower roles. A Leader that survives longer, and a cluster with more Followers retaining active energy, allow more requests to reach a majority of the cluster before the Leader itself becomes unavailable. Under the heaviest workload in Scenario 9, the relative improvement with harvesting enabled remains substantial but smaller than without harvesting, consistent with the smaller margin harvesting provides once request-driven consumption dominates the Leader's total energy expenditure. It is also worth noting that, under both harvesting conditions, the largest relative improvement occurs in Scenario 9, the combination of the largest cluster size and heaviest workload evaluated, indicating that EDRaft's advantage does not diminish under the most demanding configuration tested but instead remains most pronounced there.

5.2.2 Total Network Energy

Figures 8 and 9 present the total network energy depletion over time for all four evaluated protocols in Scenario 2 (harvesting disabled) and Scenario 9 (harvesting enabled), respectively. In both the non-harvesting and harvesting scenarios, the baseline protocols' total network energy stabilizes at an approximately constant value for extended intervals, well before the cluster's remaining capacity is exhausted. In Scenario 2, this stabilization becomes apparent from approximately 6,000s onward. In Scenario 9 (Figure 9), a comparable stabilization occurs during both unfavorable periods, near 60,000s and again near 145,000s. This behavior is consistent with the fixed-majority requirement shared by the three baseline protocols. As nodes deplete and fail, the number of live nodes eventually falls below the quorum needed not only to replicate entries but even to elect a new Leader, since a Candidate under these protocols requires the same fixed majority to win an election. Once this threshold is crossed, the cluster can neither process requests nor recover a functioning Leader, halting further energy consumption even though some nodes retain residual charge.

EDRaft's total network energy curve does not exhibit this stabilization in either scenario, declining continuously throughout the observed window instead. In Scenario 2, the curve continues to decline until approximately 8,500s, extending the cluster's operational window by roughly 2,500s, about 42% longer, beyond the point at which the baseline protocols stabilize near 6,000s. In Scenario 9, EDRaft's curve approaches, without reaching, zero during both unfavorable periods, indicating continued operation deeper into the cluster's available energy. This distinction is consistent with EDRaft's dynamic quorum mechanism, which lowers the required majority as nodes deplete, allowing the cluster to remain operative on a progressively smaller active subset instead of halting once the original majority becomes unreachable. The mechanism's floor of two active nodes sets the practical limit of this extension. The cluster can continue

Table 8. Requests Processed, Harvesting Disabled and Enabled

Scenario	Protocol	Processed Requests	
		Harvesting OFF	Harvesting ON
2	Raft	11554	247953
	EDRaft	16680	297068
	E-Raft	11569	247289
	Nakagawa	11618	248294
3	Raft	12626	378238
	EDRaft	19950	432228
	E-Raft	12579	371537
	Nakagawa	13020	380936
5	Raft	14700	276629
	EDRaft	27600	345183
	E-Raft	14400	274520
	Nakagawa	17100	279756
6	Raft	16217	422453
	EDRaft	29773	529723
	E-Raft	16302	428577
	Nakagawa	17020	426427
8	Raft	16800	296381
	EDRaft	30000	344176
	E-Raft	17200	295293
	Nakagawa	17200	300130
9	Raft	18906	454597
	EDRaft	40656	634887
	E-Raft	18521	452776
	Nakagawa	20055	459041

processing requests as long as at least two live nodes remain, rather than requiring the original fixed majority to survive intact.

Nakagawa's curve reaches its own stabilization point earlier in time than Raft and E-Raft in Scenario 2, while settling at a higher energy level. A possible explanation lies in the fraction of Followers suspended at initialization under Nakagawa's protocol. Since these nodes do not participate in message exchange while suspended, the pool of active nodes available to sustain quorum starts smaller than under the other baseline protocols, allowing the fixed-majority threshold to be crossed sooner even though the suspended nodes' own energy remains largely unused. Because a suspended node cannot initiate its own resumption, and the resume mechanism can only be triggered by an active Leader or Candidate, once every active node has depleted, the suspended nodes' energy is no longer accessible to the cluster, leaving Nakagawa to stabilize earlier, and at a higher residual energy level, than Raft and E-Raft. In Scenario 9, this pattern is visible only during the first unfavorable period (near 60,000–90,000s); no comparable elevation above the other baseline protocols appears during the second (near 140,000–180,000s), consistent with the one-time nature of this suspension mechanism, since previously suspended nodes, once resumed to restore quorum, are not suspended again and behave identically to Raft and E-Raft for the remainder of the simulation.

5.2.3 End-to-End Latency

With respect to end-to-end latency, EDRaft exhibits a substantially higher value than the three baseline protocols in every scenario evaluated. This metric captures request completion latency, the time between a client request arriving at the Leader and its response reaching the client, and is distinct from Leader failure-detection delay, which is discussed separately in Section 5.2.4. Table 9 reports the average end-to-end latency observed for each protocol across the six scenarios involving active client workload, under both harvesting conditions. In Scenario 9, for instance, EDRaft's average end-to-end latency reaches 395.6 ms, compared to approximately 7.7 to 8.6 ms for the remaining protocols, a pattern consistent with values observed in Scenarios 2, 3, and 6. This increase orig-

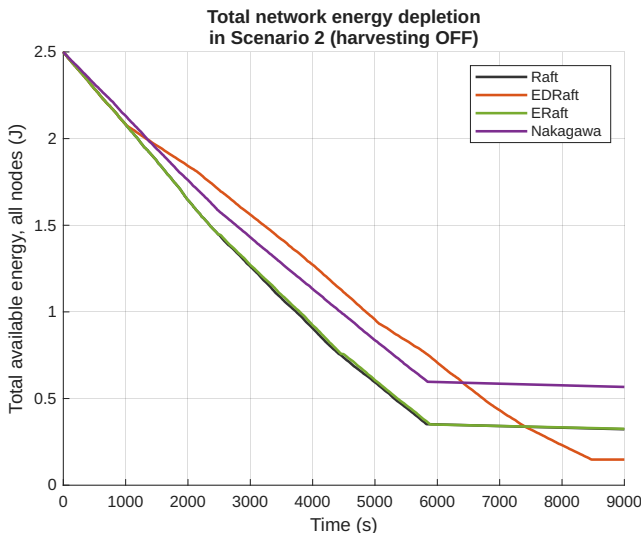


Figure 8. Total network energy depletion in Scenario 2, harvesting OFF.

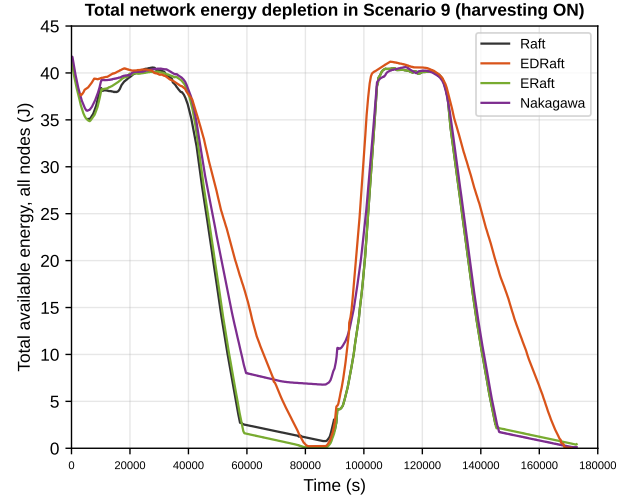


Figure 9. Total network energy depletion in Scenario 9, harvesting ON.

inates from two sources already discussed in relation to the isolated Leader and Proxy scenarios: the batching of client requests until the next scheduled heartbeat during the Attention and later phases, and the additional network hop introduced when a request is relayed through a Proxy during the Alert phase. This latency cost represents the direct counterpart to the throughput and energy improvements reported elsewhere in this evaluation, and its presence indicates that EDRaft's energy savings are obtained in exchange for slower individual request completion.

Under harvesting-enabled conditions, EDRaft's average end-to-end latency remains substantially higher than the baseline protocols in every scenario, for the same two structural reasons identified above. However, EDRaft's average latency is lower with harvesting enabled than without it in five of the six scenarios evaluated, namely Scenarios 2, 3, 5, 8, and 9, with the largest reductions observed in Scenario 8, from 375.1 ms to 238.1 ms, and Scenario 2, from 332.1 ms to 140.7 ms. This reduction is consistent with the Leader spending proportionally less time in the Alert and Critical phases when energy is periodically replenished, since these are precisely the phases responsible for Proxy relay and the most aggressive request batching; a Leader that recovers into Attention or Abundance more frequently relays fewer requests through an intermediate hop and batches them over shorter windows. Scenario 6 is the sole exception, where average latency increases slightly from 351.1 ms to 383.0 ms, suggesting that under a moderate cluster size combined with the heaviest workload, the energy recovered through harvesting is insufficient to meaningfully shift the Leader out of its most restrictive phases. The baseline protocols, by contrast, show no comparable sensitivity to harvesting, remaining within the same 7 to 8 ms range regardless of condition.

5.2.4 Fault Detection Trade-off

The heartbeat interval extension applied during the Attention phase introduces a trade-off between energy savings and fault detection latency. Because Followers extend their election timeout in proportion to the Leader's heartbeat interval, a genuine Leader failure occurring during the Attention phase or beyond takes proportionally longer to be detected than

Table 9. Average End-to-End Latency, Energy Harvesting Disabled and Enabled

Scenario	Protocol	Avg. Latency (ms)	
		Harvesting OFF	Harvesting ON
2	Raft	7.49	7.21
	EDRaft	332.05	140.74
	E-Raft	7.48	7.20
	Nakagawa	7.99	7.47
3	Raft	7.48	7.18
	EDRaft	303.74	271.73
	E-Raft	7.47	7.17
	Nakagawa	7.99	7.42
5	Raft	7.57	7.51
	EDRaft	312.49	222.72
	E-Raft	7.61	7.54
	Nakagawa	7.68	7.81
6	Raft	7.81	7.54
	EDRaft	351.07	383.00
	E-Raft	7.80	7.52
	Nakagawa	8.47	7.84
8	Raft	7.50	7.61
	EDRaft	375.12	238.06
	E-Raft	7.84	7.65
	Nakagawa	8.01	7.97
9	Raft	7.70	7.62
	EDRaft	395.60	366.69
	E-Raft	7.80	7.61
	Nakagawa	8.57	7.92

under standard Raft, delaying the start of a new election.

Since the role-isolated evaluation assigns an unlimited energy budget to every role other than the one under study, it does not, by design, capture Leader failures or the resulting re-election dynamics; the practical impact of this trade-off can therefore only be assessed under the network-level evaluation, where all roles interact and Leader failures do occur.

Across this evaluation, we find no evidence that the extended detection window degraded cluster availability. EDRaft consistently processes more client requests than every baseline protocol in every active-workload scenario, instead of fewer, as would be expected if delayed failure detection carried a meaningful cost to the cluster's ability to serve requests. This does not directly measure the time spent without an active Leader, a limitation we acknowledge and leave to future work.

6 Conclusion

This paper presented EDRaft, a phase-based, energy-aware evolution of the Raft consensus algorithm for energy-constrained environments. EDRaft enables nodes to adapt their behavior according to their remaining energy levels, optimizing consumption and extending availability.

We evaluated EDRaft against Raft, E-Raft, and Nakagawa's protocol across nine scenarios, varying cluster size, client workload, and harvesting condition. EDRaft's Leader consistently outlasted its counterpart under the baseline protocols, by 9.8 to 40.8% under active workload and up to 78.7% when idle, owing to heartbeat interval extension and Proxy delegation, with the Follower's Suspension Mode producing a similar energy-preserving effect. These role-level savings

translated into 44 to 115% higher request throughput and fewer dropped requests at the network level, an advantage that persisted with harvesting enabled, where EDRaft also exhibited reduced end-to-end latency relative to the harvesting-disabled condition in most scenarios. This resilience is reinforced by EDRaft's dynamic quorum mechanism, which keeps the cluster operative on a shrinking active subset instead of halting once the baseline protocols' fixed majority becomes unreachable.

Conversely, as anticipated, the Proxy role incurs an energy penalty compared to standard Followers, resulting from its additional coordination responsibilities, such as mediating heartbeats and forwarding client requests. This individual cost is a necessary trade-off that preserves the Leader's energy and proved beneficial for the cluster's overall survival.

However, this improvement came at a cost; EDRaft consistently exhibited higher end-to-end latency than the baseline protocols, a direct consequence of request batching and Proxy relaying. This makes EDRaft best suited to applications tolerant of delayed acknowledgment, such as supply chain tracking, decentralized energy trading microgrids, and environmental monitoring, while the trade-off would carry more weight in latency-sensitive settings, such as real-time industrial control loops, where the added delay from batching and relaying could become a limiting factor.

Future work will explore several directions building on the results presented here. First, validating EDRaft on physical hardware, or under a simulated scenario incorporating failure types beyond energy depletion, such as node crashes and network partitions, would test whether the energy savings observed here persist when node failures cannot be distinguished from temporary unreachability. Second, since the isolated Proxy evaluation revealed a substantial and consistent energy cost associated with the relay role, a natural extension is a rotation mechanism that distributes this cost across multiple Followers over time instead of repeatedly selecting the same highest-energy Follower, potentially extending overall cluster lifetime further. Third, the current evaluation assumes a homogeneous cluster, in which every node shares the same initial energy and harvesting capacity; extending the evaluation to heterogeneous deployments, where nodes differ in battery capacity or panel size, would better reflect the variability expected in real-world sensor network deployments. Fourth, while the network-level results show no degradation in request throughput attributable to the heartbeat interval extension's fault detection trade-off, this evidence is indirect, since instrumenting the cumulative time the cluster spends without an active Leader would allow this trade-off to be assessed. Finally, the Attention and Alert thresholds (60% and 30%, respectively) were set based on the rationale discussed in Section 4.2, but their values were not derived from a systematic sensitivity analysis; future work should explore how varying these thresholds affects the trade-off between early, low-cost energy-saving measures and the survival benefit of the more restrictive behaviors they trigger, potentially identifying protocol configurations tailored to specific hardware or deployment profiles.

Acknowledgements

This paper is an extension of the conference paper "da Silva Barros, G. M., de Aquino, G. S., de Queiroz Lopes, P. H., & Gonçalves de Oliveira, F. T. (2023). EDraft: An energy-aware consensus algorithm for low-power IoT devices. In 2023 Symposium on Internet of Things (SIoT) (pp. 1–5). IEEE. <https://doi.org/10.1109/SIoT60039.2023.10390014>".

Availability of data and materials

Data sharing or additional materials is not applicable to this article.

Funding

This work was carried out with the support of the Coordination for the Improvement of Higher Education Personnel – Brazil (CAPES) – Financing Code 001

Authors' Contributions

GB was responsible for the writing process of the paper, as well as for the conception, implementation, and experimental evaluation of the proposed protocol. GA contributed to the conception of the protocol and writing process. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

References

- Balsamo, D., Weddell, A. S., Merrett, G. V., Al-Hashimi, B. M., Brunelli, D., and Benini, L. (2015). Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embedded Systems Letters*, 7(1):15–18. DOI: 10.1109/LES.2014.2371494.
- Bhat, A., Bandrupalli, A., Nagaraj, M., Bagchi, S., Kate, A., and Reiter, M. K. (2023). Eesmr: Energy efficient bft—smr for the masses. In *Proceedings of the 24th International Middleware Conference*, Middleware '23, page 1–14, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3590140.3592848.
- da Silva Barros, G. M., de Aquino, G. S., de Queiroz Lopes, P. H., and Gonçalves de Oliveira, F. T. (2023). EDraft: An energy-aware consensus algorithm for low-power IoT devices. In *2023 Symposium on Internet of Things (SIoT)*, pages 1–5. DOI: 10.1109/SIoT60039.2023.10390014.
- Fernandez Blanco, D., Le Mouël, F., Ponge, J., and Lin, T. (2023). An Energy-efficient FaaS Edge Computing platform over IoT Nodes: Focus on Consensus Algorithm. In *Symposium on Applied Computing (SAC)*, Tallinn, Estonia. ACM/SIGAPP. DOI: 10.1145/3555776.3577598.
- Fletcher, B. J., Balsamo, D., and Merrett, G. V. (2017). Power neutral performance scaling for energy harvesting mp-socs. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2017, pages 1516–1521. DOI: 10.23919/DATE.2017.7927231.
- Heinzelman, W. R., Chandrakasan, A., and Balakrishnan, H. (2000). Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences (HICSS)*, pages 1–10, Maui, HI, USA. IEEE. DOI: 10.1109/HICSS.2000.926982.
- Jayakumar, H., Raha, A., and Raghunathan, V. (2014). Quick-recall: A low overhead hw/sw approach for enabling computations across power cycles in transiently powered computers. In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, pages 330–335. DOI: 10.1109/VLSID.2014.63.
- Lamport, L. (2001). Paxos made simple. *ACM SIGACT News (Distributed Computing Column)* 32, 4 (Whole Number 121, December 2001), pages 51–58. Available at: <https://www.microsoft.com/en-us/research/publication/paxos-made-simple/>.
- Li, X., Guo, Y., Yan, L., and Xu, X. (2021). Energy-aware blockchain for multiple autonomous underwater vehicles cooperative operation. In *2021 40th Chinese Control Conference (CCC)*, pages 3005–3010. DOI: 10.23919/CCC52363.2021.9549537.
- Lucia, B., Balaji, V., Colin, A., Maeng, K., and Ruppel, E. (2017). Intermittent Computing: Challenges and Opportunities. In Lerner, B. S., Bodik, R., and Krishnamurthi, S., editors, *2nd Summit on Advances in Programming Languages (SNAPL 2017)*, volume 71 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:14, Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. DOI: 10.4230/LIPIcs.SNAPL.2017.8.
- Merrett, G. V. and Al-Hashimi, B. M. (2017). Energy-driven computing: Rethinking the design of energy harvesting systems. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2017, pages 960–965. DOI: 10.23919/DATE.2017.7927130.
- Nakagawa, T. and Hayashibara, N. (2018). Energy efficient raft consensus algorithm. In Barolli, L., Enokido, T., and Takizawa, M., editors, *Advances in Network-Based Information Systems*, pages 719–727, Cham. Springer International Publishing. DOI: 10.1007/978-3-319-65521-5_64.
- Ongaro, D. and Ousterhout, J. (2014). In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, Philadelphia, PA. USENIX Association. Available at: <https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro>.
- Pâris, J.-F. and Long, D. D. E. (2015). Pirogue, a lighter dynamic version of the raft distributed consensus algorithm. In *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8. DOI: 10.1109/PCCC.2015.7410281.
- Shetty, S. S., Kamhoua, C. A., and Njilla, L. L. (2019). *Distributed Consensus Protocols and Algorithms*, pages 25–50. DOI: 10.1002/9781119519621.ch2.
- Sliper, S. T., Cetinkaya, O., Weddell, A. S., Al-Hashimi, B., and Merrett, G. V. (2019). Energy-driven computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2164):20190158. DOI: 10.1098/rsta.2019.0158.

Tuncel, Y., Bhat, G., Park, J., and Ogras, U. Y. (2022). Eco: Enabling energy-neutral iot devices through runtime allocation of harvested energy. *IEEE Internet of Things Journal*, 9(7):4833–4848. DOI: 10.1109/JIOT.2021.3106283.