Computer Science Technical Report

A Practical Self-Stabilizing Leader Election for Networks of Resource-Constrained IoT Devices

Michael Conard and Ali Ebnenasir

Michigan Technological University
Computer Science Technical Report
CS-TR-21-01
April 2021

Department of Computer Science
Houghton, MI 49931-1295
www.cs.mtu.edu

A Practical Self-Stabilizing Leader Election for Networks of Resource-Constrained IoT Devices

Michael Conard and Ali Ebnenasir

April 2021

Abstract. This paper presents a self-stabilizing and tunable leader election algorithm, called PraSLE, designed for real-world testbeds of resource-constrained devices in the Internet of Things (IoT). PraSLE is an extended version of the minimum finding algorithm that functions in a round-based asynchronous fashion. We provide a version of PraSLE for both reliable and unreliable networks. We show that PraSLE elects a unique leader in unreliable networks with probability 1. We also experimentally validate this result and demonstrate that PraSLE outperforms existing methods in terms of average convergence time for ring, line, and mesh topologies up to 40 nodes, and for the clique topology up to 80 nodes. The tunable nature of PraSLE enables engineers to optimize average convergence time, as well as communication and energy costs. PraSLE provides an important resource-efficient Distributed Computing Primitive (DCP) for unreliable networks of low-end IoT devices based on which other DCPs (e.g., Paxos) can be developed for such networks in the domains of IoT and Cyber Physical Systems (CPS).

Keywords. Leader Election, Self-stabilization, Resource-constrained Devices, IoT

Contents

1 Introduction

The development of Distributed Computing Primitives (DCPs) for resource-constrained distributed systems is an important challenge in the Internet of Things (IoT) and Cyber Physical Systems (CPSs) applications. Examples of such DCPs include leader election, consensus, spanning tree formation, distributed reset, etc. Developing lightweight DCPs that can be deployed in real-world unreliable networks (e.g., UDP/IP) is a challenging problem. More importantly, such networks are subject to transient faults such as bad initialization and loss of coordination, and they are expected to correctly execute DCPs despite the presence of such faults. One of the important distributed algorithms that has applications in numerous settings (e.g., broadcast, Paxos [16]) includes the Leader Election (LE) algorithm. While there are numerous algorithms developed for LE, it is desirable to have a resource-efficient, scalable, and self-stabilizing implementation of LE that can provide a *sub-linear* average convergence time on unreliable networks and in the presence of transient faults.

Numerous approaches exist for leader election in IoT and networks of resource-constrained devices, most of which validate their protocol through simulation and lack evaluations on a real-world testbed. For example, Méndez *et al.* [18] implement the Bully algorithm [12] on a clique network where the process that has the maximum identifier (ID) becomes the leader. However, they provide little experimental results on a physical testbed. The Minimum Finding (MinFind) algorithm [21] extends the idea of the Bully algorithm, where each node broadcasts a randomly generated value (or its ID) and then enters a cycle of reading and relaying the minimum value through broadcast. Bounceur *et al.* [3] present a Wait-Before-Send (WBS) protocol for synchronous networks, where each node waits for $min \times t$ seconds before sending out its min value. Bounceur *et al.* [2] also present an energy-efficient algorithm for LE, Branch Optima to Global Optimum (BROGO), where a spanning tree is formed by flooding rooted at a specific node whose ID is known to everyone. Sheshashayee *et al.* [23] put forward a protocol for electing the top k leaders in an asynchronous clique network with a broadcast primitive, and evaluate it through simulation and on a testbed of just 11 nodes.

Most aforementioned approaches are evaluated through simulation or validation on small-scale testbeds of less than 30 nodes. Moreover, it is unclear if the underlying network is a UDP-based network. Some existing methods assume a synchronous network. For example, the WBS method requires that all processes are synchronized at the start of the protocol, which is hard to ensure in large-scale distributed systems. Some approaches are asymmetric in that the code of some nodes are different from the rest of the nodes. Asymmetric LE algorithms are harder to implement and are more costly in terms of communication and energy (e.g., BROGO [2]). Another restrictive assumption for a practical real-world LE implementation includes the topology; where most existing methods assume a clique topology, we provide an efficient, tunable implementation and demonstrate it on ring, line, tree, mesh, and clique topologies. Last but not least, our experiments show that PraSLE can easily scale up to networks of about 80 nodes with close to sub-linear increase in average convergence time, which is an important property for practical use in IoT and CPSs.

Contributions. First, we present a practical, self-stabilizing and tunable version of the MinFind algorithm, called PraSLE[1], that takes two parameters: maximum network latency and a diameter-proportional value. Mainstream engineers can use these parameters as knobs for tuning PraSLE to their platform, thereby striking a balance between robustness and resource efficiency. Second, our algorithm is self-stabilizing in that it can recover to electing a unique leader from transient perturbations to arbitrary states. We provide variants of PraSLE for both reliable and unreliable networks. We prove the correctness of PraSLE, and show that it converges with probability 1 (in unreliable networks). Third, in contrast with most existing methods, we actually implement our algorithm (instead of just simulating it) on the RIOT operating system [1] in networks of up to 80 machines with Cortex-M3 processors in the IoT-Lab (`www.iot-lab.info`). Fourth, PraSLE is symmetric in that all nodes have the same piece of code up to variable renaming. Symmetry is an important feature for ease of use and energy consumption. Fifth, our communication model is timed asynchronous, where we have no synchronization assumption. We evaluate PraSLE using the unicast primitive on a UDP/IP

[1]Implementation source code available at `https://github.com/maconard/cps-iot_leader-election`

network (see Section 4). Sixth, while most existing methods consider a clique topology, we evaluate PraSLE on ring, line, tree, mesh, and clique topologies. Our experiments show that leader election on mesh is the most resilient and time, communication, and energy-efficient election. This is an interesting result for settings where engineers have the liberty of configuring their desired topology. Finally, we experimentally evaluate PraSLE with respect to three metrics: average convergence time, communication costs, and energy costs. Our experiments show that PraSLE's average convergence time grows very slowly like a linear function (in networks of up to 80 nodes) with a tiny slope that varies from 0.02 to 0.04 for different topologies. The average number of messages exchanged grows linearly for all topologies (except clique), but significantly slower for line and ring. The energy consumption of the entire network grows like a step function for line and ring with a tiny step of 0.05J. For mesh and clique, energy costs grow linearly but with a tiny slope of 0.04 for up to 40 nodes. Once the network scales beyond 40 nodes in clique the slope of the linear growth becomes 0.07. We note that we created these topologies as overlay networks, and achieving such resource efficiency on these overlay networks using a unicast primitive is a significant contribution by itself. In summary, our experiments show that PraSLE is a resilient, scalable, and resource-efficient leader election protocol for real-world networks of low-end devices. To the best of our knowledge, no other implementation of leader election exhibits such characteristics on a wide range of topologies and in unreliable UDP networks.

Organization. Section 2 states the LE problem. Section 3 present the PraSLE algorithm along with its proof of correctness. Section 4 provides our experimental results. Section 5 expands on our experimental results in regards to the tunable values of PraSLE. Section 6 discusses related work. Finally, Section 7 makes concluding remarks and discusses future extensions of this work.

2 Problem Statement

This section states the problem of Leader Election (LE) in a distributed system as well as presenting the practical requirements under which we should solve the LE problem.

> *Given is a set of computing nodes/processes connected through a communication network. Does there exist an algorithm deployed on each process such that all processes collectively elect a unique process as the leader without reliance on any centralized authority*?

While there are numerous methods for solving the LE problem under different settings (see Section 6), our objective is to provide a simple solution that is both self-stabilizing and can be deployed on practical networks (e.g., UDP/IP) for a large number of nodes. Thus, we first discuss the computation and communication models under which we solve the LE problem.

Computation model. The processes are *symmetric* in that the codes of any pair of processes are similar up to variable renaming/re-indexing. Each process p_i has a unique identifier ID_i, which can simply be its assigned IPv6 address. Each instruction/action of the algorithm has a finite execution time. Actions are *read-write atomic*; that is, each read or write action is considered atomic. For example, checking the truth of a condition may not necessarily be atomic, but it contains atomic actions for reading the components of the condition. Each process has a *ranking value*, denoted min_i (e.g., Line 4 in Figure 1), that defines the potential of that process for becoming the leader. The lower the value of min_i, the higher the likelihood of p_i becoming the leader. Such a ranking value could be randomly generated or may be a function (e.g., getRankingValue() in Line 5 of Figure 1) of quantitative metrics such as battery life, processing power, available memory, workload, etc. The variable $leader_i$ in each process p_i (e.g., Line 6 in Figure 1) holds the ID of the process that p_i believes to be the leader.

Communication model. The communication model of our algorithm is *timed asynchronous*, where messages may be delivered by a maximum delay known by all nodes (denoted T in Figure 1). We consider both reliable and unreliable networks, where messages may be lost. The underlying network topology determines the neighborhood of each process.

Fault and failure models. The value of min_i (representing the ranking function) may be perturbed by transient faults, which could result in the perturbation of the global state of the system. Examples of transient faults include soft errors (which cause random bit flips in memory), bad initialization, and loss of

coordination. Processes may also fail in a detectable manner (using a time out mechanism), and messages could get lost in communication.

3 The PraSLE Algorithm

This section presents a self-stabilizing solution for the LE problem where starting from any arbitrary initial values of min_i, the entire network will eventually elect a unique leader whose identity will be known for each process. Since our main objective is to design an algorithm that can be implemented and deployed on real-world large-scale networks (e.g., the Internet), we call it Practical Self-stabilizing Leader Election (PraSLE). We also derive a probabilistic version of the proposed algorithm that will elect a leader with probability 1 in unreliable networks.

We first discuss a deterministic version of PraSLE where we assume reliable communication (i.e., messages are delivered up to an upper bound delay known by each process). Figure 1 illustrates the pseudo code of a representative process in PraSLE. Each process executes in a round-based fashion and the rounds in different processes need not be synchronized. However, we require that from the time a node sends a message until the message is received, the sender can perform a finite number of rounds. The algorithm runs for *at least* K rounds (Line 2 in Figure 1), where K can initially be the diameter of the network, denoted D_{LE}, but could be initialized with other values too. In the first round, the condition of the `while` loop is false (because *round* is equal to $K+1$) and each process sends its min_i and $leader_i$ values to its immediate neighbors (Lines 20 to 26). Starting from the second round, all processes go through repetitive execution of three main tasks: (1) collect information from neighbors for T seconds (Lines 11 to 18); (2) update local knowledge about who the leader is (Lines 20 to 22), and (3) disseminate local knowledge to immediate neighbors (Lines 23 to 25). Note that any node can initiate the election and the nodes will propagate that event to their neighbors; this snippet is excluded from Figure 1 for brevity.

Task 1 is to listen for T seconds to receive new information from neighbors which was sent out in the conclusion of the previous round (i.e.; the previous Task 3). When a message is received by process p_i from some neighbor p_j, process p_i updates its local information with min_j and $leader_j$ if the pair $(min_j, leader_j)$ is smaller than the pair $(temp\_min_i, temp\_leader_i)$ (see Lines 14-15 in Figure 1). We say that $(m, ID) < (m', ID')$ *if and only if (iff)* $(m < m') \vee ((m = m') \wedge (ID < ID'))$. Even if process p_i receives a message from all of its neighbors it will still wait the full T seconds of the round duration before moving on.

Task 2 is simply a comparison between the most recent values of the leader and its ID in the current round with the best values of the previous round. If the values of the current round (saved in the pair $(temp\_min_i, temp\_leader_i)$) are a better choice then the pair $(min_i, leader_i)$ is updated (in Lines 21 and 22) to those values. Otherwise, no changes are made.

Task 3 is the dissemination of new best leader information to immediate neighbors. This can be implemented with a broadcast or uni-cast primitive. In a naive implementation the nodes can send their best choice of leader to their neighbors every round regardless of whether it is new information or not. However, a simple optimization is to have nodes track which neighbors already know the information they have and exclude them from the update, which significantly improves message complexity. This improvement is not depicted in Algorithm 1 for simplicity, but we have implemented it in the code used for experimentation.

3.1 Correctness

This section defines the important requirements of PraSLE and then proves the correctness of the algorithm under reliable and unreliable network assumptions. First, we define the *improvement* requirement as follows:

Definition 3.1 (Improvement)**.** Once a process receives the ID of the global leader it will never lose it. Thus, once a unique leader is identified and every node has received its ID, it remains the leader of everyone.

In order to formalize the notion of improvement, we define the function $C(s)$ from the set of global states of the algorithm to $\{0, 1, \cdots, N-1\}$, where a global state is a global snapshot of the values of variables of all processes:

$$C(s) = N - |\{\text{nodes that have found } leader \text{ in state } s\}|$$

Observe that, $C(s)$ is a non-increasing function because the number of nodes that have found the leader only increases (or remains unchanged). For PraSLE to find a unique leader, the value of $C(s)$ must eventually become zero. That is, all nodes will eventually receive the ID of the leader.

Algorithm 1: Practical Self-Stabilizing Leader Election

```
process p_i;                                                              (1)
  var  round := K + 1; // K can be the network diameter                   (2)
       neighbors_i := getListOfNeighbors(); // Neighbor ID list           (3)
       min_i := N + 1; // N is the max number of processes                (4)
       temp_min_i := getRankingValue();                                   (5)
       leader_i := ID_i; // Identifier of process p_i                     (6)
       temp_leader_i := ID_i;                                             (7)
       T := 1.0; // A tunable value                                       (8)

begin until False                                                         (9)
  Timer recvTimer := T;                                                   (10)
  while recvTimer > 0 and round < K + 1 {                                 (11)
      :: recv (min_j, leader_j) from p_j in neighbors_i {                 (12)
          if (min_j, leader_j) < (temp_min_i, temp_leader_i) {            (13)
              temp_min_i = min_j;                                         (14)
              temp_leader_i = leader_j;                                   (15)
          }                                                               (16)
      }                                                                   (17)
  }                                                                       (18)
  round − −;                                                              (19)
  if (temp_min_i, temp_leader_i) < (min_i, leader_i) {                    (20)
      min_i = temp_min_i;                                                 (21)
      leader_i = temp_leader_i;                                           (22)
      for p_j in neighbors_i {                                            (23)
          send(p_j, min_i, leader_i)                                      (24)
      }                                                                   (25)
  }                                                                       (26)
  else if round <= 0 → return leader_i                                    (27)

end                                                                       (28)
// In case of unreliable networks, Lines 23 to 25 are moved outside
// the 'if statement' of Line 20, and Line 27 is removed
```

Figure 1: The proposed Leader Election algorithm.

Definition 3.2 (Convergence)**.** Each node will eventually receive the ID of the leader. That is, the value of $C(s)$ eventually becomes zero.

Notice that, *Improvement* is a safety property; i.e., something bad (e.g., losing the ID of the leader) never happens in any computation of the algorithm. The *Convergence* property is a liveness property; i.e.,

something good will eventually occur in every computation of the algorithm.

Correctness in reliable networks. First, we prove the correctness of the algorithm (with respect to improvement and convergence) under the assumption of a reliable network, where messages are delivered in a finite amount of time and no messages are lost.

Theorem 3.3. *It is always the case that, starting from any state, PraSLE meets the improvement requirement. Formally,* $\Box(C(s_i) \geq C(s_{i+1}))$*, where* $\Box$ *denotes the 'always' temporal operator, and* $(C(s) \geq C(s+1))$ *states that* $C(s)$ *remains non-increasing when transitioning from a state* s_i *to its subsequent state* s_{i+1}*.*

Proof. We will show that PraSLE satisfies the Improvement property via an induction on an arbitrary computation. Let c be a computation $< s_0, s_1, ..., s_m >$ of length m which starts in state s_0 and where (s_i,s_{i+1}) is a state transition for $i \geq 0$. We model the global state transition system of the algorithm as a non-deterministic finite state machine, where each state is a global snapshot of the local state of each node, and each transition captures message transmission, receipt of a message, or executing a local instruction on any node. Note that only Lines 21-22 of the algorithm are capable of impacting the value of C.

Base case. In the initial state s_0, $C(s_0) \geq C(s_1)$ holds because at the start of the algorithm it is impossible to update min_i and $leader_i$, and messages are not sent or received yet.

Inductive hypothesis. Let $C(s_{k-1}) \geq C(s_k)$, for $k \geq 2$.

Inductive step. We must consider the different possible state transitions that could be performed from any state s_k and show that $C(s_k) \geq C(s_{k+1})$ must hold. If the state transition (s_k,s_{k+1}) is the execution of any machine instruction other than those from Lines 21-22, then the value of C is unchanged and the inductive hypothesis holds as $C(s_k) = C(s_{k+1})$. If (s_k, s_{k+1}) is the execution of the instructions from Lines 21-22, then because of the condition on Line 20 the information being stored to min_i and $leader_i$ must be better than the previous values. If these new values are not the global leader's then C remains unchanged; i.e., $C(s_k) = C(s_{k+1})$. If these new values are the global leader's then C is decremented by one and $C(s_k) > C(s_{k+1})$. Since the network forms a connected graph, at least one node must find the leader and decrement C this way in every round of the algorithm. There is no case where the value of C is increased. Therefore, we have $C(s_k) \geq C(s_{k+1})$; i.e., PraSLE satisfies the Improvement property.

□

Theorem 3.4 (Convergence)**.** *PraSLE will eventually elect a unique leader. That is,* $\Diamond(C(s) = 0)$ *holds, where* $\Diamond$ *denotes the 'eventually' temporal operator.*

Proof. Similar to the proof of *Theorem 1*, at least one node will obtain the global leader's information in each round which will reduce the value of C by at least 1. In the worst case, when the global leader is D_{LE} hops away from some node in the network, that node will receive the information in the D_{LE}-th round. Since C was initialized with $N-1$ (a finite value) and the network forms a connected graph, C must reach 0 in finite time. Thus, the global state of the election must proceed closer to a unanimous decision where C converges to 0; i.e., PraSLE satisfies the Convergence property. □

Correctness in unreliable networks. In the case of unreliable networks, we create a revised version of PraSLE by moving Lines 23 to 25 outside the 'if statement' of Line 20, and by removing Line 27. For simplicity, we consider a worst case reliability $0 < \alpha < 1$ for the network. That is, the probability of a message being delivered from a sender to a receiver is at least α. Note that when $\alpha = 1$, we have a reliable network, and when $\alpha = 0$, the network is totally broken. For example, when a process p_i executes a 'recv' action in Line 12 of Figure 1, the message is received from p_j with at least a probability α. Likewise, the 'send' action in Line 24 would succeed in delivering the min_i and $leader_i$ to p_j with at least a probability α. We follow the technique in [13] and show that Improvement and Convergence hold with probability 1.

Theorem 3.5. *Let* s_0 *be a state where* $C(s_0) = L$ *for some* $0 < L < N$*.* [2] *We show that in any computation of the form* $\langle s_0, s_1, s_2, \cdots \rangle$*, there is some* $k > 0$ *where* $C(s_k) = L-1$ *with probability 1. Formally,* $((C(s_0) = L) \wedge (0 < L < N)) \implies Pr(\exists k : k > 0 : (C(s_k) = L-1)) = 1$.

[2] We assume $L < N$ because there is at least one process that has the minimum ranking value.

Proof. First, observe that $\Pr(\exists k : k > 0 : (C(s_k) = L - 1)) = 1 - \Pr(\forall k : k > 0 : (C(s_k) = L))$. We show that $\Pr(\forall k : k > 0 : (C(s_k) = L)) = 0$; hence, $\Pr(\exists k : k > 0 : (C(s_k) = L - 1)) = 1$.

Let p_i be the process that has the minimum ranking value globally. As such, p_i sends min_i and $leader_i$ to its immediate neighbors. Let Δ denote the max degree of the underlying network topology. That is, p_i can have at most Δ immediate neighbors. The probability that a single neighbor does not receive min_i and $leader_i$ is at most $1 - \alpha$, and the probability of none of the neighbors receiving min_i and $leader_i$ is equal to $(1 - \alpha)^{\Delta}$. Thus, the probability of the immediate neighbors of p_i never receiving the values would be $\lim_{k\to\infty}(1-\alpha)^{k\Delta}$. Since $(1-\alpha) < 1$, we have that $\lim_{k\to\infty}(1-\alpha)^{k\Delta} = 0$. Thus, a neighbor must eventually receive the values after some finite k steps of computation with probability 1 and at that time we will have $C(s_k) = L - 1$. This implies that $\Pr(\forall k : k > 0 : (C(s_k) = L)) = 0$ and based on the above observation we get $\Pr(\exists k : k > 0 : (C(s_k) = L - 1)) = 1$ given $(C(s_0) = L) \wedge (0 < L < N)$. Thus, the revised PraSLE satisfies the Improvement property in unreliable networks with probablity 1. □

Theorem 3.6. *Let s_0 be a state where $C(s_0) = L$ for some $0 < L \leq N$. We show that in any computation of the form $\langle s_0, s_1, s_2, \cdots \rangle$, there is some $k > 0$ where $C(s_k) = 0$ with probability 1. Formally, $((C(s_0) = L) \wedge (0 < L \leq N)) \implies Pr(\exists k : k > 0 : (C(s_k) = 0)) = 1$.*

Proof. From some state s_0 with $C(s_0) = L$ and $(0 < L < N)$ the Improvement property gives that $\Pr(\exists k : k > 0 : (C(s_k) = L-1)) = 1$, where k is a finite number of computational steps. The Improvement property can be repeated for several rounds subsequently, each having another finite k steps of computation, until L reaches 0. Since the initial value of L is finite, the total number of rounds will also be finite. Each round completes with probability 1, thus having $\Pr(\exists k : k > 0 : (C(s_k) = 0)) = 1$. Thus, the revised PraSLE satisfies the Convergence property in unreliable networks with probability 1. □

Theorem 3.7. *In reliable networks, the worst case convergence time of PraSLE is $O(D)$, where D is the network diameter.*

Proof. PraSLE has two phases: one includes Lines 11 to 18 where a node waits for T seconds to receive updates from its immediate neighbors, and the second phase contains Lines 20 to 26, where each node P_k sends out local updates to its immediate neighbors. In a reliable network, we are sure that the neighbors of P_k will receive P_k's updates. Now, let P_l be the node whose initial min_l is the global minimum. In the first round of dissemination of min_l the immediate neighbors of P_l receive min_l and update their local min values. The immediate neighbors of P_l will disseminate min_l to their neighbors in their next round. Continuing thus, it will take at most the diameter hops away from P_l until every node has min_l and the ID of P_l. Therefore, the convergence time to electing a leader is proportional to the diameter of the network; i.e., $O(D)$. □

Theorem 3.8. *In reliable networks, the worst case message complexity of PraSLE is $O(D\Delta N)$, where D represents the network diameter, N is the number of nodes, and Δ denotes the max degree of the network.*

Proof. In each round of message exchange between a node P_k and its immediate neighbors, P_k receives at most Δ message by executing Lines 11 to 18. Then, P_k may send at most Δ messages to its immediate neighbors in Lines 20 to 26. Theorem 3.7 shows that each node will execute at most $O(D)$ rounds. Thus, in the worst case, each node will exchange $O(D\Delta)$ messages. Since we have N nodes in the network, the overall worst case message complexity of PraSLE is $O(D\Delta N)$. □

4 Experimental Results

This section presents our results regarding the experimental evaluation of PraSLE on the ring, line, mesh, tree and clique topologies. We consider three major performance metrics: average convergence time, average number of messages exchange and average energy consumption. We perform the experimental evaluation of PraSLE on the IoT-Lab (`www.iot-lab.info`) system with iotlab-M3 nodes, which are physical IoT devices comprised of Cortex M3 32-bit CPUs with a maximum clock speed of 72 MHz. These nodes have 64KB of

RAM and 256KB of ROM and communicate using the IEEE 802.15.4 standard in the 2.4GHZ frequency band. Their maximum bandwidth is 256 kbits/s and they have a range of about 50 meters indoors. The communication model is asynchronous and unreliable (UDP), which means there are delayed and lost packets. The energy monitoring is provided by the IoT-Lab platform via an INA226 hardware component with a sampling period of 65.95 ms, which has maximum error of 0.1%.

We evaluate the reliable form of PraSLE, despite the fact that it is an unreliable environment. Each topology is evaluated for a range of the tuneable values K and T over the different network sizes. The data for every configuration of K, T, and N is an average of 10 executions of PraSLE, or fewer for some of the more aggressive configurations of K and T with a low probability of succeeding, particularly in larger networks. In the context of running the reliable form of PraSLE in an unreliable network, aggressive K and T values are those closest to or more strict than the optimal reliable values. Section 5 investigates the impact of the values of K and T on convergence time, messages, and energy consumption. Instances that fail due to the nature of an unreliable network are disregarded to collect results on instances that succeed, as if they were in a reliable network. That is, running the reliable version of PraSLE in an unreliable network does not always succeed, and a new instance is launched until 10 successful results or 90 total minutes have passed. The graphs in Figures 2, 3, and 4 illustrate how increasing the size of the network in each topology impacts those same metrics. The tree topology is depicted separately in Figure 5.

Ring topology. We have evaluated the ring topology for 10 to 40 nodes (see left side of Figures 2, 3, and 4). Since a ring is broken with only two failed nodes, it was not practical to evaluate more than 40 nodes on the IoT-Lab as the odds of such a failure becomes too great. However, each of the segmented regions of the network under such a failure do still elect a leader. The diameter of a ring network is $N/2$ so it follows that the optimal K value is also $N/2$, which indeed gave the best results for smaller network sizes. Larger network sizes feel the impact of the unreliable network more and required more robust K values, such as the ring of size 40 which began succeeding at K = 23 rather than the optimal K = 20. This is because when the packets containing the global leader's information are dropped, that round results in no progress. In an unreliable network of a larger size and small average degree this is bound to happen some times, so additional rounds are needed as a buffer to account for this.

Line topology. We have evaluated the line topology for 10 to 30 nodes (see left side of Figures 2, 3, and 4). Since a line is broken with only one failed node, which makes it even more likely to break than a ring, we could not evaluate networks greater than 30 nodes on the IoT-Lab platform. Similarly to the ring, each of the segmented regions of the network under such a failure do still elect a leader. The diameter of a line network is N so it follows that the optimal K value is also N, although we test more aggressive K values as well. These more aggressive tests with K less than the optimal value succeed with much lower frequency, but still provide valuable data points for approximating growth curves of the different metrics we consider.

Mesh topology. We have evaluated the mesh topology for 10 to 40 nodes (see right side of Figures 2, 3, and 4). The mesh topology has an average degree greater than three (approaching four as N grows), so it exhibits significant robustness compared to line and ring topologies. Network sizes greater than 40 were not evaluated only because of the uncertainty of availability regarding running experiments on the shared IoT-Lab platform. The diameter of a mesh network is $L + W - 2$ where L is the length (or height) of the network and W is the width of the network. For a mesh as square as possible, the values are $L = \lceil \sqrt{N} \rceil$ and $W = round(\sqrt{N})$; e.g. $L = 7$ and $W = 6$ for a mesh of size 40, which is six complete rows of six and one partial row of four.

Clique topology. We have evaluated the clique topology for 10 to 80 nodes (see right side of Figures 2, 3, and 4). This topology can be thought of as a complete graph; it has an average degree of N-1, so it exhibits significant robustness compared to the other topologies at the expense of an increased communication cost. Network sizes greater than 80 were not evaluated because of memory constraints on the low resource devices. We are currently working on optimizing the use of memory so we can scale up beyond 80 nodes. The diameter of a complete network is exactly one, which means the optimal K value is one. However, due to the large degree of the network there are a significant number of messages exchanged, so a slightly larger K such as two or three allows for improved robustness in unreliable networks with only a small impact to time.

Tree topology. We have evaluated the tree topology for 10 to 40 nodes (Figure 5), for only one K and

T valuation. The binary tree network is constructed as complete as possible, where the layers of the tree are filled in left to right. The height of a binary tree of size N is $O(\ln N)$, so the diameter of the network (through the root node) is also $O(\ln N)$. Note that these results are with a conservative T value.

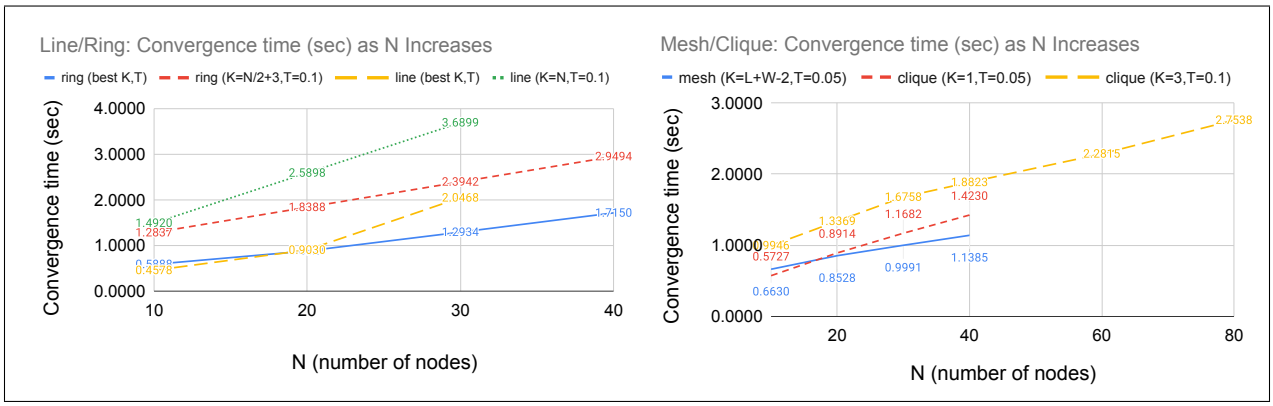

Figure 2: Convergence time (sec) as network size grows

Figure 2 shows that the convergence time grows along a linear curve for the ring and line topologies. The number of data points is relatively small, so the curves for mesh and clique do not represent their asymptotic time complexity well, as they also appear linear. The curves marked "(best K,T)" are made from the best K and T pair available for each network size, which is why the yellow/long-dash curve in the left graph juts upward after 20, as more aggressive parameters were used at N of 10 and 20 than at 30 for the line topology.

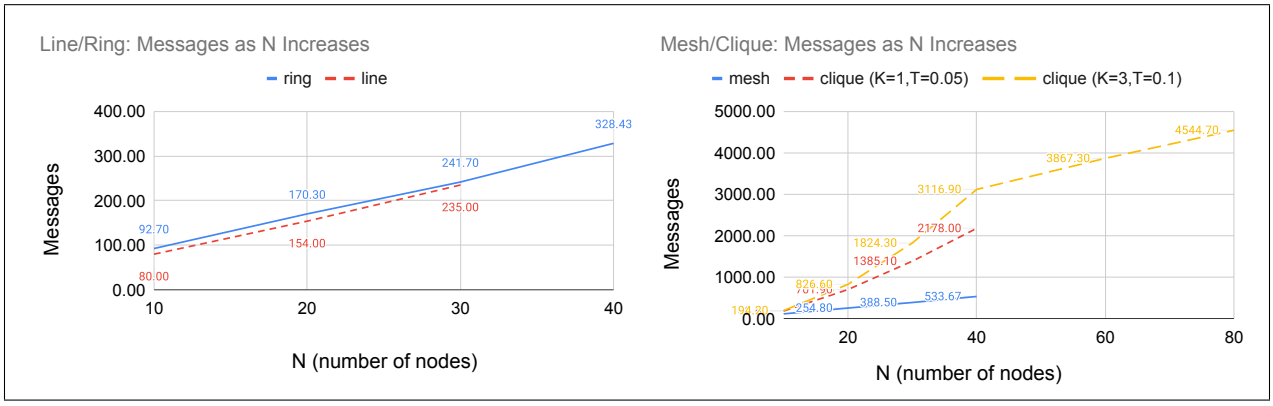

Figure 3: Communication costs as network size grows

Figure 3 illustrates the growth in communication costs as the size of the network increases. The ring and line topologies shown in the left graph are similar, as the average degree is two for a ring and approaches two as the network grows for a line. The mesh topology's average degree approaches four as the network grows, so the growth is similar to ring and line but with a greater constant factor. The clique topology's average degree is N-1, so the worst case number of messages is $N * (N-1) * K$, which is quadratic unlike the other topologies which are linear. After N=40 the clique switches to linear growth because the nodes have reached the maximum number of messages they can process within the round duration. Note that the clique topology would benefit the most from a broadcast primitive but it is evaluated with unicast for consistency with the other topologies.

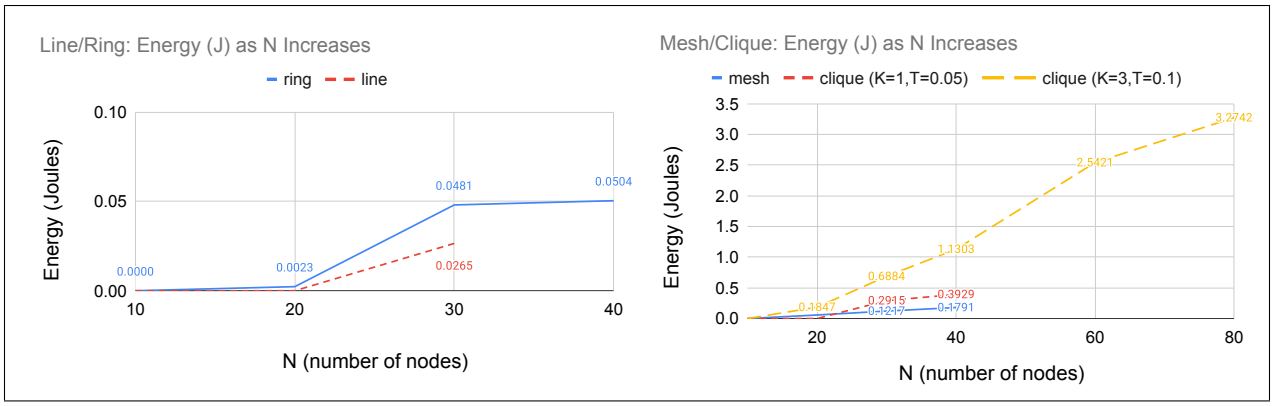

Figure 4: Energy (J) costs as network size grows

Figure 4 presents the energy consumption of PraSLE as the network size grows. It is expected that this should be some combination of convergence time and message cost, as they are the two largest contributors to total energy consumption. Some of the smaller sizes of the topologies are shown at 0 because energy consumption with PraSLE running was not differentiable from the energy consumption of idle nodes, on average. The ring and line topologies are incredibly efficient in terms of energy given their low message cost. The clique topology consumes a significantly larger amount of energy as the network grows since the message cost explodes (see Figure 3). The best middle ground is the mesh topology, which converges quickly, has a reasonable message cost and energy consumption, and has increased reliability due to its higher average degree.

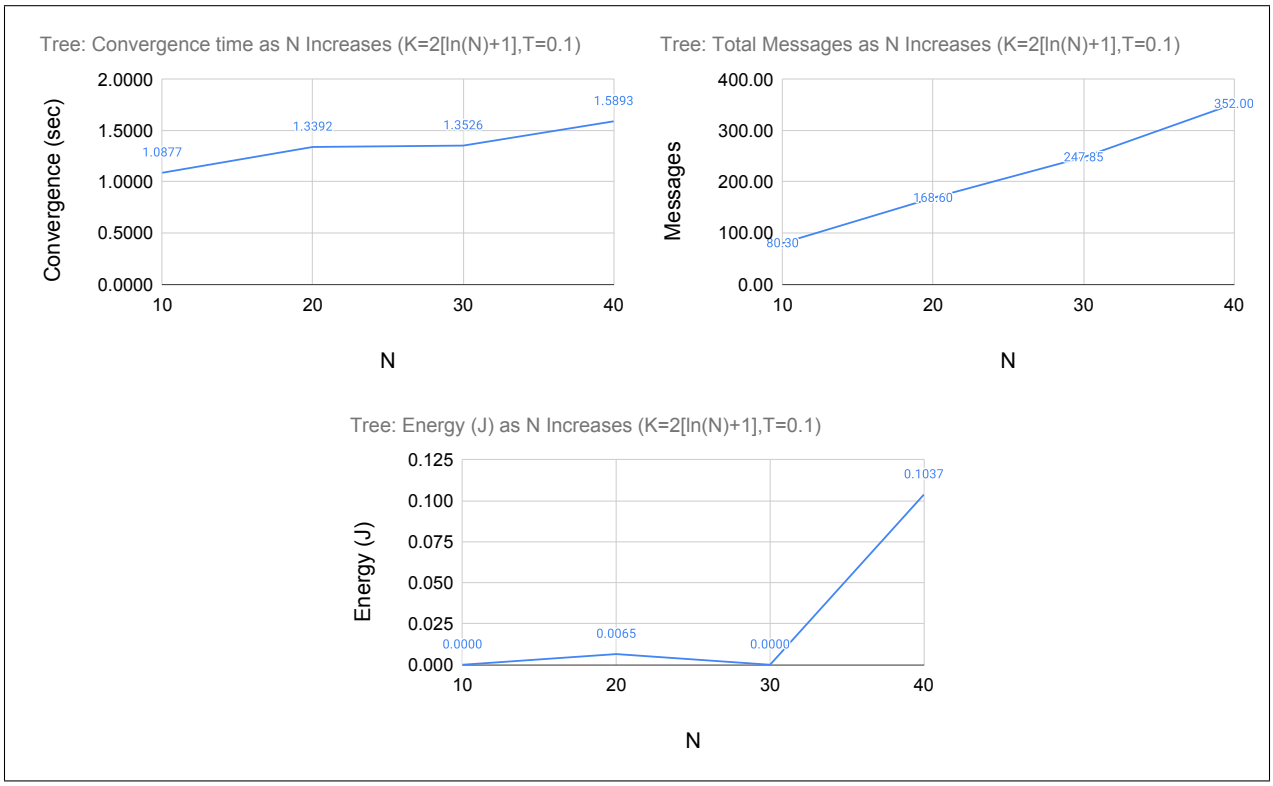

Figure 5: Average convergence time, communication costs, and energy costs for tree as network size grows

Figure 5 shows the convergence time, communication costs, and energy costs for a binary tree network. Since binary trees of size 20 and 30 have the same height, we expect their convergence times to be nearly identical, which is reflected in the left figure. The message cost increases linearly with the number of nodes, also as expected. This demonstrates how when the diameter of the network grows poly-logarithmically, then the convergence time complexity of PraSLE is also poly-logarithmic. The energy costs are close to 0 J for sizes 10, 20, and 30, while size 40 consumed 0.1 J on average. These results were collected with a conservative T=0.1.

Summary of results. From our theoretical results we expect PraSLE to grow in convergence time relative to how the diameter of the network grows. Experimentally, we find that PraSLE exhibits slow linear growth with a small constant factor across the topologies, although we expect that with a significantly larger evaluation scope we could validate sub-linear growth on the non-linear topologies. The clique is a special case; its network diameter never increases from one so theoretically neither should the convergence time, but practically the number of messages exchanged with a unicast primitive grows rapidly. As a result, the convergence time grows with an incredibly small constant factor along with the message cost increase. More precisely, PraSLE's average convergence time grows very slowly like a linear function (in the number of nodes) with a slope of less than 0.04 for line and ring, and with a slope of less than 0.02 for tree, mesh, and clique topologies. The average number of messages exchanged grows linearly for all topologies (except clique), but significantly slower for line and ring. The energy consumption of the entire network grows like a step function for line and ring with a tiny step of 0.05J. For mesh and clique, energy costs grow linearly but with a tiny slope of 0.04 for up to 40 nodes. Once the network scales beyond 40 nodes the slope of the linear growth becomes 0.07.

For all topologies the convergence time of PraSLE is around half a second for a network of size 10 (with the best K and T values attempted) and they remain under two seconds for networks of size 40. In a network with maximum message delay of less than 0.05 seconds these times could be further improved. The growth in communication cost of PraSLE is linear when $\Delta \ll N$, with the constant factor varying by the average degree of the topology. On topologies like line and ring with average degrees of 1.8 to 2, PraSLE takes 10s to 100s of messages as the network grows, while topologies like mesh and clique with average degrees around 3.5 and N-1 respectively take hundreds to thousands as the network grows. That is, since the clique's average degree is N-1 it effectively has a quadratic message cost. The message costs recorded in the data and figures is the sum of incoming and outgoing messages at each node, so received messages are double counted. This allows for dropped or lost messages to be factored in to the cost. Note that because these experiments are on overlay topologies, a broadcast primitive would break the overlay and contact neighbors the node should not have. The energy consumption of PraSLE is directly related to the number of messages exchanged as well as the duration of the election. Ring and line topologies consume only hundredths of a Joule, the mesh topology consumes tenths of a Joule, and a size 80 clique consumes up to 3 Joules. For some small networks of size 10 and 20 across the topologies the energy consumption is negligible such that it could not be differentiated from a set of idle nodes with the accuracy provided by the IoT-Lab energy monitoring system. Note that the graphed energy consumption data are adjusted for the energy consumption of idle nodes to more accurately reflect the energy cost of PraSLE specifically. The average energy consumption per node can be computed by dividing the energy consumption by the size of the network; this version of the data is excluded for brevity.

Comparison with the state-of-the-art. It is difficult to compare PraSLE with related work due to different hardware platforms and experimental conditions. However, given the unreliable nature of UDP/IP and the use of the unicast primitive (instead of broadcast), PraSLE provides unmatched performance with respect to related work (Section 6 presents a thorough discussion). For example, WiLE [23] performs most of their experimental evaluation via simulation, but they do provide a small evaluation on an 11 node testbed of irregular topology, with average degree 4.0. The closest match to that from our evaluation of PraSLE is the 10 node mesh topology. In this case WiLE outperforms in terms of convergence time (0.17 to 0.66), however, it is unclear how WiLe would behave if it were deployed on an unreliable UDP network. Energy costs appear to be similar; WiLE gives an average 0.006 J energy cost, while PraSLE's energy cost was not measurable for this small topology, given that it could not be distinguished from idle nodes (i.e.; it is similarly close to

0). Lastly, PraSLE outperforms in messaging costs. Recall that our figures double count packets as outgoing messages and incoming messages (to reflect lost packets), so PraSLE exchanged approximately 55-60 packets on average while WiLE exchanged 103.7 packets on average. Of course, it is worth noting that WiLE and PraSLE are evaluated on entirely different hardware. Bully LE [18] is evaluated on Cortex-M3 hardware similarly to PraSLE, although with only 6 nodes. Unfortunately, as far as we are aware, the authors of [18] did not provide any convergence time, message cost, nor energy cost data to compare to. Resilient cluster LE [9] is evaluated on TelosB nodes in a cluster of size 50. Only the nodes above a certain energy threshold compete to become the leader, so not all 50 nodes are eligible like with PraSLE. The authors of [9] focus on security and do not provide any comparable metrics in terms of convergence time, messaging cost, and energy cost, although they do state that their communication cost is "not a big problem."

5 Tuning K and T

The values K and T used in PraSLE are tunable in that they can be raised and lowered to impact robustness (in an unreliable network) and efficiency. Recall that K is the number of rounds the protocol will run for and T is the duration of each round in seconds. Since the location of the global leader within the network cannot be known in advance, K must be at least the diameter of the network. It can be increased further to account for dropped packets where the global leader's information fails to propagate in some rounds. Increasing the value of K has a small impact on convergence time and energy consumed, so it is worth the exchange for added reliability in a practical environment. The value of T should be as small as possible while allowing for essentially all messages to be delivered in time. By testing a range of values we found that the value of 0.05 seconds for T consistently gives fast convergence in Iot-Lab while allowing most messages to be successfully delivered in time, assuming they are not lost completely, for all topologies. A more aggressive T than 0.05 may be possible on other network configurations. Increasing the value of T has a large impact on convergence time and therefore energy consumed, for the benefit of giving more time for messages to reach their destination. Neither increasing K nor T has any significant impact on the message complexity, since nodes only send messages as needed in the first place. The best initial configuration to tune from is setting K to the diameter of the network and T to 0.05, increasing K by one until the desired reliability is achieved. Since increasing T has drastic impacts on time efficiency it is not recommended unless the communication delay of the network is significant, in which case fast convergence would already be challenging. As the size of the network grows so does the number of lost messages, so more robust K and T values may be needed. An example of such a case is the ring of size 40, where the best K value was 23 rather than the theoretically optimal 20. Of course, in a reliable communication environment K should be set exactly to the diameter of the network and T should be set exactly to the network's maximum message delay.

The graphs in this section portray the impacts of changing K and T on convergence time, communication, and energy costs on networks of fixed size. This is performed for line, ring, mesh, and clique networks of size 10, 20, and 30. The ring, mesh, and clique networks are also evaluated at size 40.

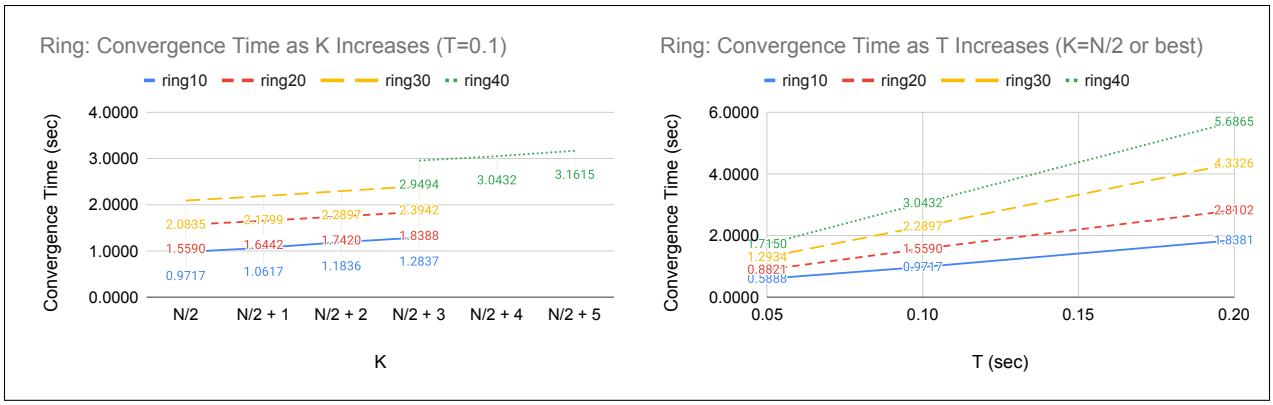

Figure 6: Average convergence time for ring as K and T change

Figure 6 illustrates the linear impact of increasing K or T in the ring network. The slope of the curve from increasing K is much less than the slope from increasing T. For example, in the N=40 ring network the cost of increasing K by 1 is approximately 0.1 seconds while the cost of increasing T by 0.05 is approximately 1.3 seconds.

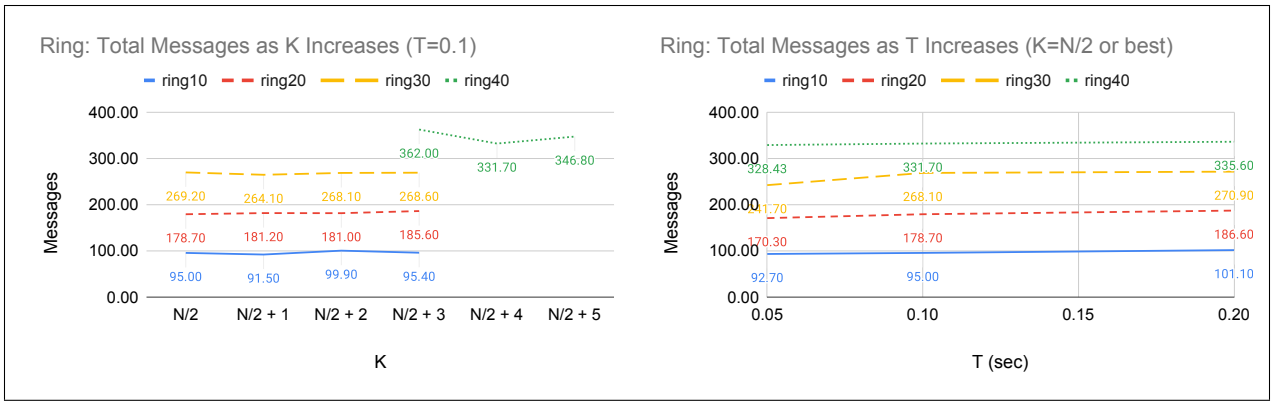

Figure 7: Communication costs for ring as K and T change

Figure 7 shows how the values of K and T do not have any particular impact on message costs in the ring network. This is because messages are only sent as needed in the first place, so neither redundant rounds nor extra time cause any additional messages.

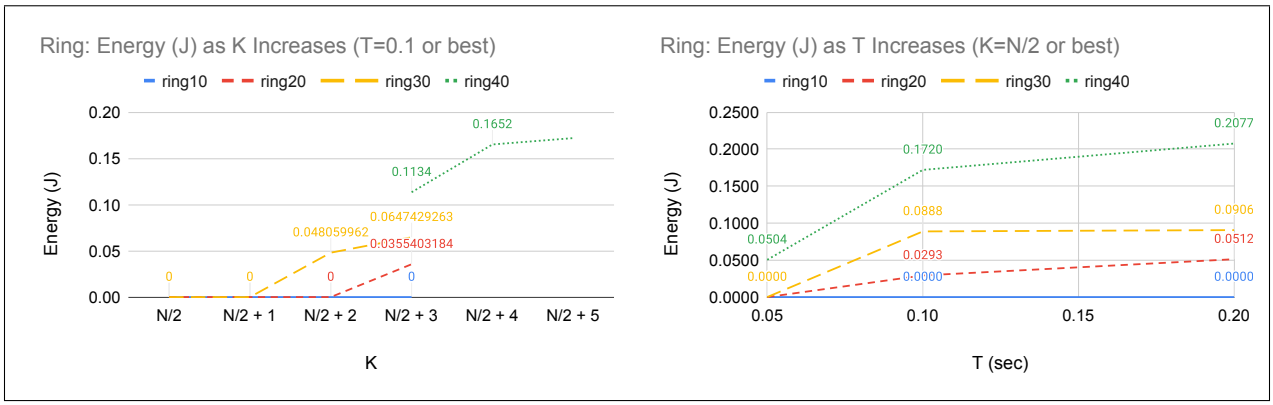

Figure 8: Energy (J) costs for ring as K and T change

Figure 8 illustrates the energy consumption of the protocol across the entire network. As mentioned earlier in this section, some of the smaller network sizes showed a negligible consumption compared to idle nodes, so they are marked as 0 Joules. The experiments that consumed a non-negligible amount of energy show a linear increase in consumption, which is related to the convergence time and message cost.

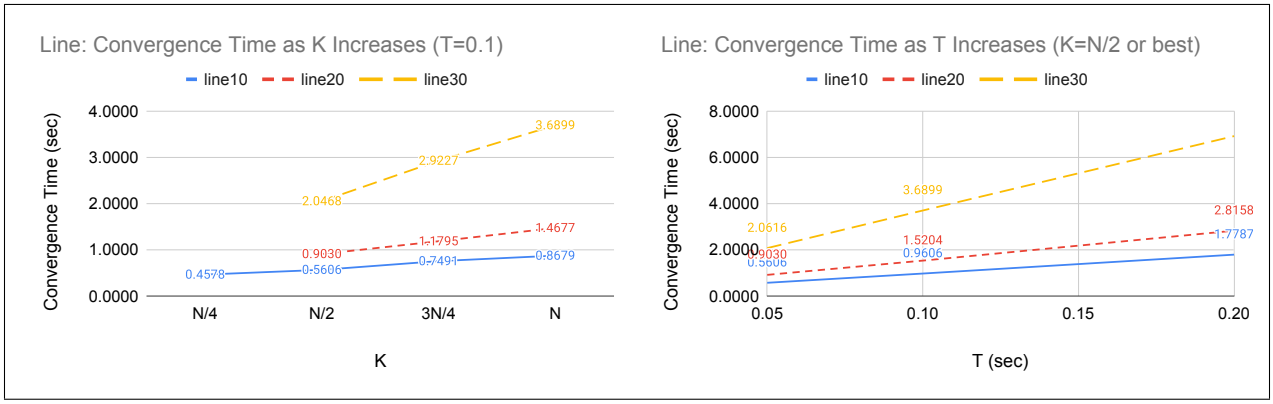

Figure 9: Average convergence time for line as K and T change

Figure 9 gives similar results to the ring topology (Figure 6), where increases in T have a much larger impact on convergence time than K in a line topology. For example, in the N=30 line network the cost of increasing K by N/4 is approximately 0.83 seconds while the cost of increasing T by 0.05 is approximately 1.64 seconds.

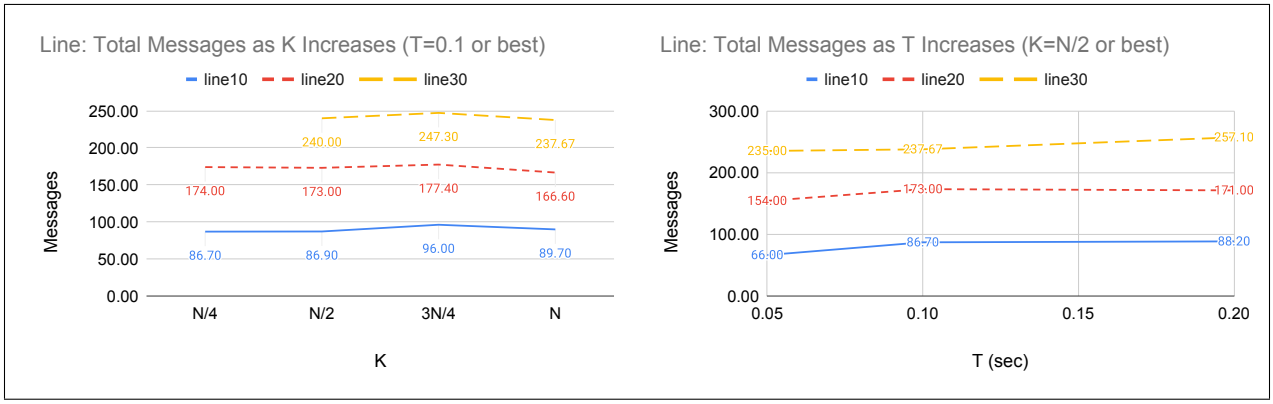

Figure 10: Communication costs for line as K and T change

Similarly to the ring topology, Figure 10 shows that increases in K and T do not have any particular impact on the number of messages exchanged.

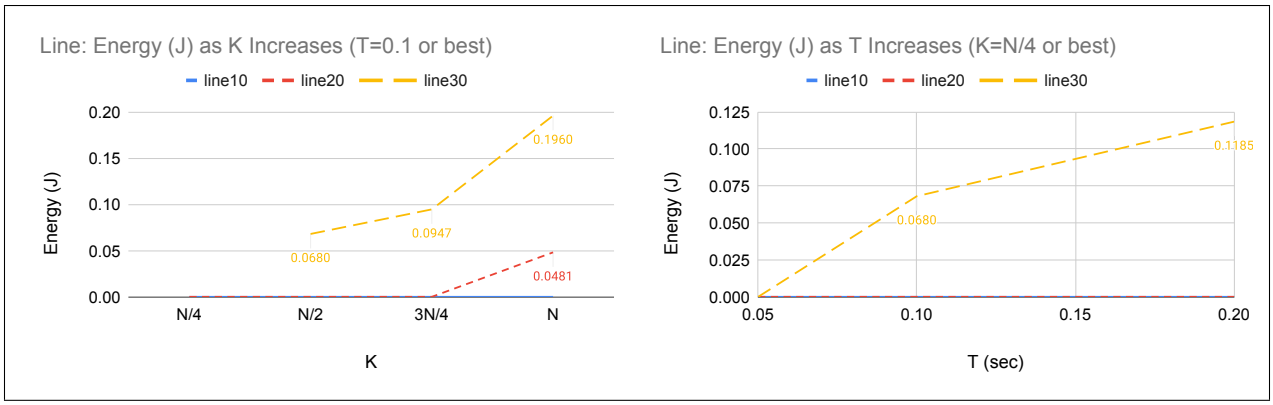

Figure 11: Energy (J) costs for line as K and T change

Figure 11 shows how little energy the line network consumes, such that it was only properly measurable for the N=30 size. Increases in both K and T had little impact on energy.

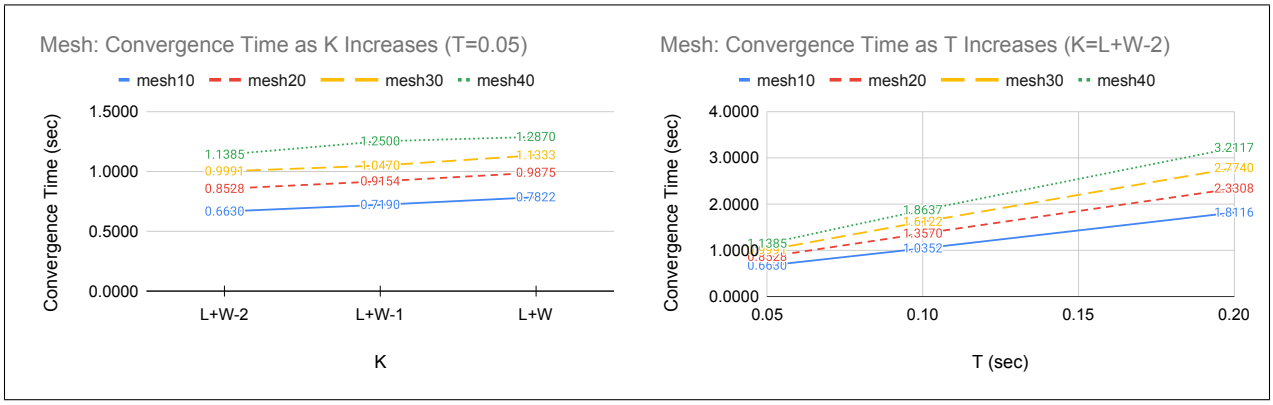

Figure 12: Average convergence time for mesh as K and T change

Figure 12 illustrates the efficiency of the mesh network. The linear (or better) growth in convergence time from increases in K and T has an incredibly small constant factor, so increases in the parameters for added reliability are not too costly. The increase in convergence time from adding one to K is only 0.07 seconds, which is from the additional round duration of 0.05 seconds and some other minor delay factors.

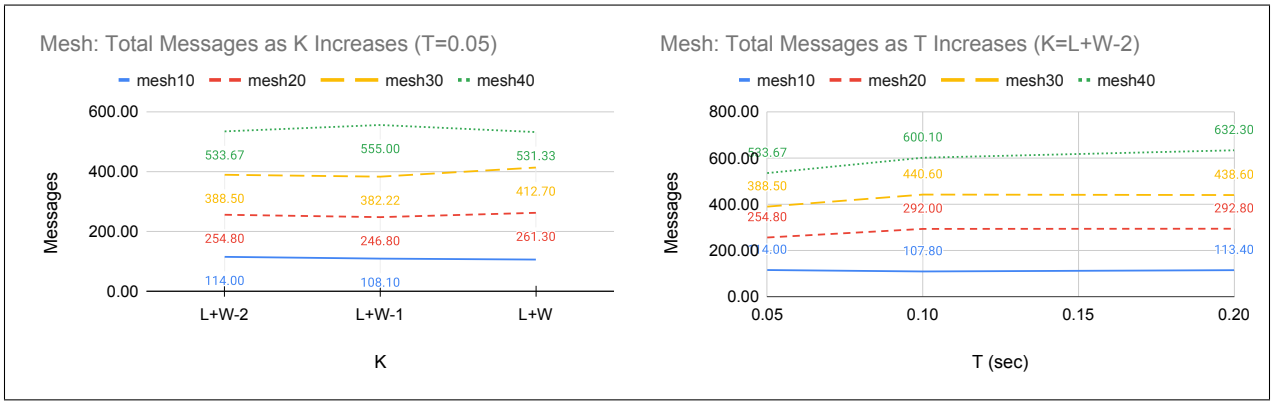

Figure 13: Communication costs for mesh as K and T change

As with the other topologies, Figure 13 further demonstrates how changes in K and T have little impact on communication cost.

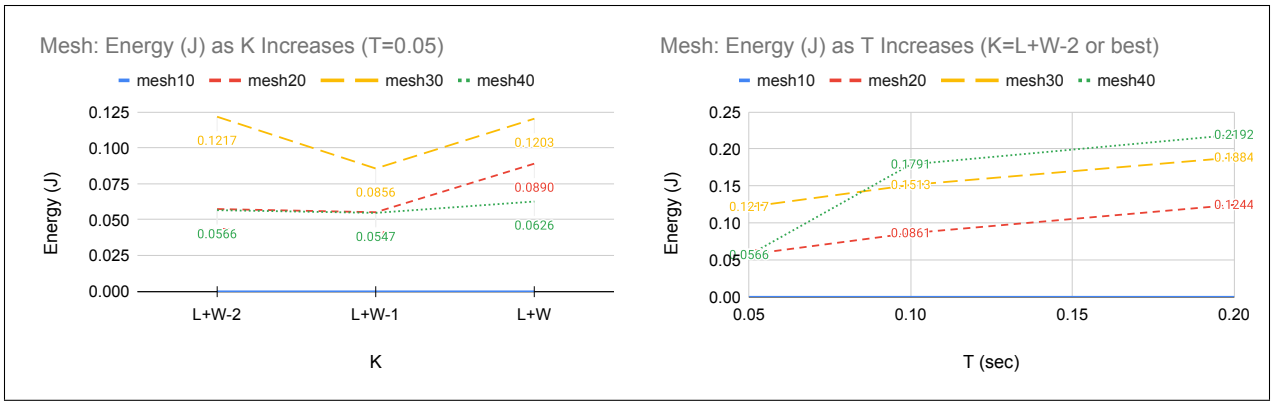

Figure 14: Energy (J) costs for mesh as K and T change

Figure 14 shows the energy consumption of PraSLE in a mesh topology. The mesh30 curve in the left figure contains a measuring error, as there is no reasonable way for a lower K to consume so much more energy. Similarly in the right figure, it is a measuring error that the mesh40 curve is initially below the mesh30 curve, although the trend corrects itself. We believe this is because with the smaller experiments or those that last only a fraction of a second, the energy monitoring does not as accurately reflect the true cost.

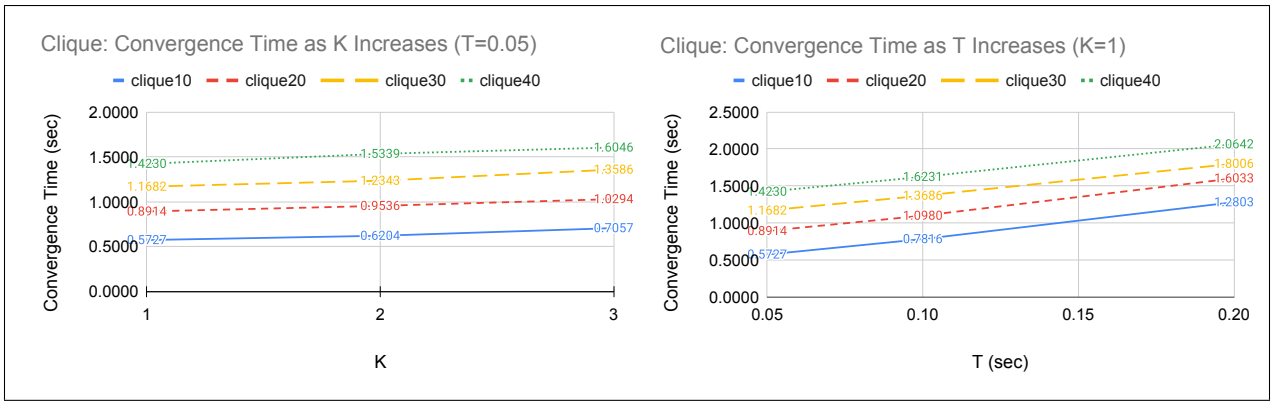

Figure 15: Average convergence time for clique as K and T change

The clique convergence times in Figure 15 appear similar to the mesh curves from Figure 12, with similar slopes. Considering the average degree of N-1, each node has a large number of messages to process each round, and much of this processing time bleeds over the final round. This implies that the best average degree for a network to have would be somewhere closer to the the mesh's 3.5-4 than the clique's N-1.

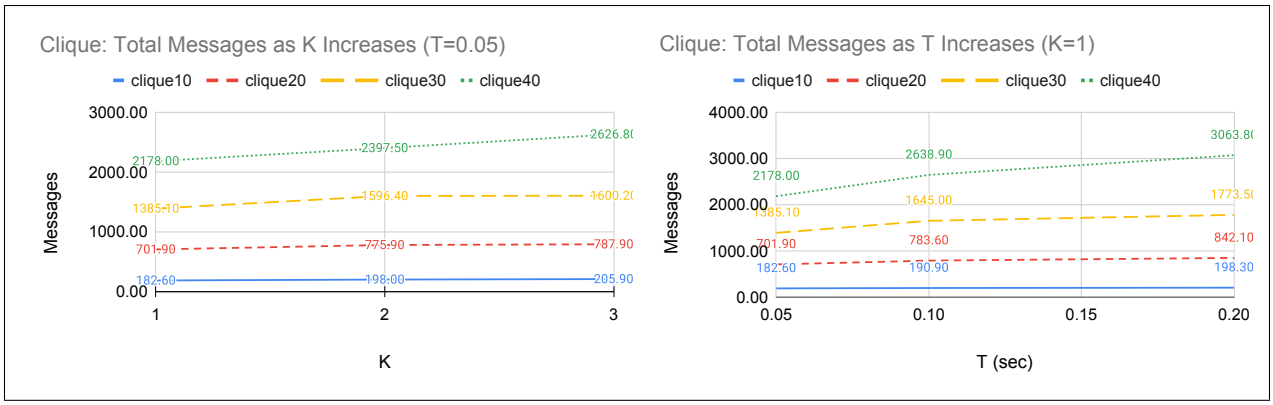

Figure 16: Communication costs for clique as K and T change

Figure 16 shows just how large the message cost is for a clique network, given that it's high average degree brings the unicast messaging cost from linear to quadratic. Unlike the other topologies where K and T do not have any noticeable impact on communication cost, with the clique there is a slight up-trend in communication cost following an increase in K or T. This is because there are so many messages being sent, a longer election allows for more of them to be received and processed.

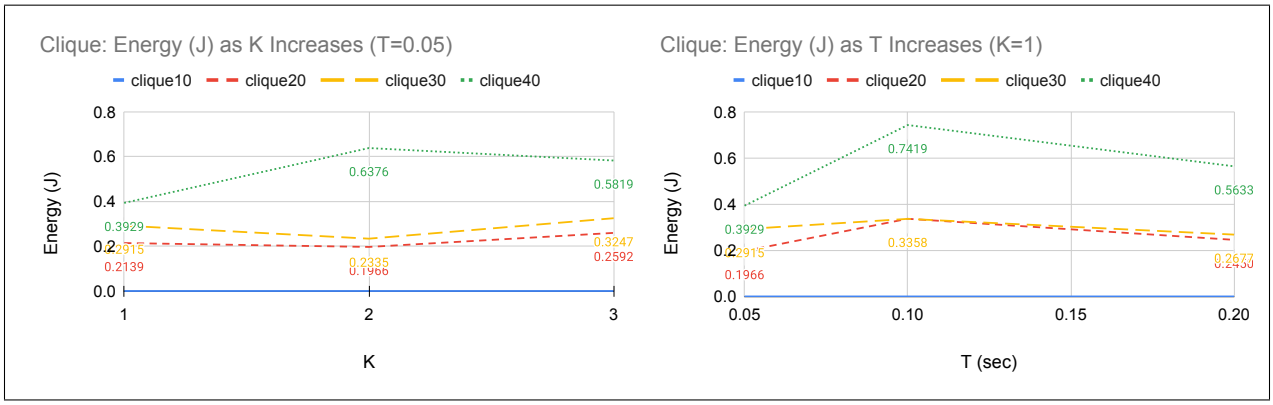

Figure 17: Energy (J) costs for clique as K and T change

Figure 16 illustrates the energy consumption of the clique topology. The size 10 clique wasn't distinguishable from idle nodes. The outliers of K=2 and T=0.1 appear to be measuring errors, as there is not an explainable reason for them to cost more than higher K and T values.

6 Related Work

This section discusses the strengths and weaknesses of existing methods while considering our research objectives of developing a practical self-stabilizing LE algorithm that can be deployed on UDP networks with a unicast primitive. Specifically, we divide the literature into research that (i) explores the theoretical boundaries of LE; (ii) approaches for LE in Wireless Sensor Networks and MANET, and (iii) studies focused on LE in IoT and resource-constrained networks.

There are numerous methods that study LE from a theoretical point of view using different computation and communication models without being concerned with actual implementation issues. For example, Doty and Soloveichik [10] present a population protocol for LE on a network of symmetric processes/nodes connected in a clique topology. They show that linear parallel time (in the number of nodes n) is a lower bound for the convergence of constant-space population protocols. Sudo *et al.* [26] improve this lower bound to

Protocol	Time Complexity	Message Complexity	Self-Stabilizing	Topology	Scale Devices	Communication Model
[23]	Polynomial	Polynomial	Unstated	Irregular ($\Delta = 4$)	11 nodes MagoNode++	Reliable Asynchronous (TCP)
[18]	Unstated	Unstated	Unstated	Clique	6 nodes Cortex-M3	Unstated
[9]	Unstated	Unstated	Unstated	Unstated	50 node cluster TelosB	Unstated
PraSLE	O(D)	O(DΔN)	Yes	Ring, Line, Tree, Mesh, Clique	40 to 80 nodes Cortex-M3	Unreliable Asynchronous (UDP)

Figure 18: Comparison with the IoT-based methods that experiment on practical testbeds.

logarithmic by relaxing the assumption of constant-space processes allowing each process to have $O(\log n)$ states. Casteigts *et al.* [7] present an LE protocol for networks of nodes with unique IDs of size $O(\log n)$ that converges in $O(D + \log n)$ bit rounds where D denotes the network diameter. The bit round complexity [15] is a measure of complexity that takes into account both the number of rounds (i.e., time) and the amount of information exchanged in a protocol until it terminates/stabilizes. Chang *et al.* [8] investigate the energy complexity of three important distributed algorithms, namely leader election, approximate counting, and census. Sidik *et al.* [24] design LE protocols under different synchrony assumptions for dynamic networks where nodes can join and leave. However, their protocols have quadratic message complexity and $nlog(n^2)$ space complexity bits per node, where n represents the number of nodes. Srinivasan and Kandukoori [25] give a modified Paxos (Progressive-Paxos) algorithm, which like Paxos depends on having a predetermined set of the nodes designated as Acceptors. Paxos is a consensus algorithm, which they directly apply to leader election. They claim their P-Paxos algorithm is scalable to large systems in the hundreds or thousands of processes and fault tolerant up to half the Acceptors failing, but their experimental work is simulated via many processes on one machine rather than on distinct low-resource or IoT devices. Their message complexity is polynomial in the size of the Acceptor set and they don't provide a time or energy complexity.

Many researchers have worked on the design and implementation of LE protocols that can be deployed in WSN and MANET. For example, Dong and Liu [9] present a round-based protocol for secure election of cluster heads in WSN and recovery in the presence of jamming, where short-term disruptions of communication occur. Richa *et al.* [20] present a self-stabilizing LE protocol for WSN where nodes communicate over a shared medium, and the proposed protocol tolerates jamming. Vasudevan *et al.* [27] present a non-stabilizing LE protocol for adhoc and mobile wireless networks based on a Diffusing Computation (DC) and spanning tree formation. Lee *et al.* [17] present a multi-leader LE protocol for minimizing the number of executions of the LE protocol in energy-constrained systems. They keep a list of 5 leaders in order of priority, thereby significantly decreasing the re-election time. Sharma and Sing [22] present an algorithm for mobile ad hoc networks where they elect a President (first leader), a Leader, and a Vice Leader. The election algorithm is run on a subset of network nodes, called the House, that includes nodes whose energy/compute power is between 70 to 100%. Zhang *et al.* [28] present an LE protocol based on proactive broadcast in Vehicular Ad-hoc Networks (VANET) and a built-in ordering function, where each node initially assumes that it is the leader and broadcasts a leader message, and waits for a specific period of time. If no messages are received, then the node continues to assume itself as the leader and broadcasts a leader message with incremental sequence number. Otherwise, it uses a built-in ordering function on received messages to compare the candidates and select a leader. However, this protocol suffers from high communication costs.

Numerous approaches exist for leader election in IoT and networks of resource-constrained devices, most of which rely solely on simulation as the method of evaluation. For example, Rahman [19] surveys existing methods for leader election in IoT and points out the fact that dependability issues (e.g., fault tolerance and security aspects) are vital properties that need to be implemented efficiently. Méndez *et al.* [18] present an implementation of the Bully algorithm [12], where leader election is performed based on the process IDs and

the process that has the max ID becomes the leader. Méndez *et al.* implement the Bully algorithm on a network with the clique topology where each node is an Arm Cortex-M3 managed by the FreeRTOS operating system, however, their work lacks experimental results on how robust and efficient their implementation is. The Minimum Finding (MinFind) algorithm [21] extends the idea of the Bully algorithm, where each node broadcasts a randomly generated value (or its ID) and then enters a cycle of reading and relaying the minimum value (i.e., flooding the minimum value) through broadcast. Bounceur *et al.* [3] present a protocol for synchronous networks based on the MinFind algorithm [21], where each node waits for $m \times t$ seconds, and if it receives nothing, then it broadcasts its m value and halts, called Wait-Before-Send (WBS). Otherwise, it broadcasts the value it has received and then halts. Bounceur *et al.* [2] also present an energy-efficient algorithm for LE, called Branch Optima to Global Optimum (BROGO), where a spanning tree is formed by flooding rooted at a specific node whose ID is known to everyone. To address the issue of the failure of the root, the authors of [4] combine BROGO with WBS in order to automatically select a different root if the primary one fails. Another work by Bounceur *et al.* [5, 6, 14] extends the logic of the MinFind algorithm where the network is initially a forest and a tree routing protocol is used to flood the network with the ID of the leader of each tree (i.e., its local optima found by MinFind). Faika *et al.* [11] where they propose a leader election protocol for the Battery Management System (BMS) in electrical vehicles, where the ranking function represents the eligibility of node for becoming a leader based on its battery level. Nodes broadcast the ranking value and wait for response. Moreover, nodes gradually increase their RF range if they do not receive any responses. After receiving the response of its neighboring nodes, each node elects a local leader. The local leaders follow the same procedure to elect a global leader.

The closest work to ours includes Sheshashayee *et al.* [23] where they put forward a protocol for electing the top k leaders, and evaluate it through simulation and on a testbed. In their LE protocol, each node has a rank between 1 and n, where n denotes the number of nodes in the network, a weight, and an ID. The communication primitive is a broadcast operation in an asynchronous clique network and the authors consider a time limit up to which messages are delivered. Initially, each node sets its rank to 1 and broadcasts a message containing its rank, its weight (which is a random value generated only once) and its ID. Then, the nodes engage in exchanging their local views, called *progress view*, of their locality and increase their rank depending on the node weight and their progress view. The objective is to eventually have the progress view of the entire network in every node where the ranks are distinct. Reaching such a state ensures that every node in the network can sort all nodes based on their rank, and as a result know who the top $k < n$ leaders are. The authors validate their method on the GreenCastalia simulator and implement and deploy it on real testbed of 11 MagoNode++ nodes, each having an 8-bit 16MHz CPU with 256 KB of ROM and 32 KB of RAM, and communicating using an IEEE 802.15.4 compliant 2.4 GHz radio module. Their simulation and experimental results show that the convergence time and communication costs are reasonable. They also show that packet size has an inverse relation with convergence time and energy consumption.

To the best of our knowledge, all aforementioned approaches are evaluated through simulation or validation on small-scale testbeds of less than 30 nodes, with the exception of [9]. Moreover, it is unclear if the underlying networks are UDP-based. The table in Figure 18 summarizes the related works that perform experimentation on real-world platforms. Some existing methods make assumptions about synchrony. For example, the WBS method [3] requires that all processes are synchronized at the start of the protocol, which is hard to ensure in large scale distributed systems. Some approaches are asymmetric, which makes it hard to implement and more costly in terms of communication and energy (e.g., BROGO [2]). Another restrictive assumption for a practical real-world LE implementation includes the topology, where most existing methods assume a clique topology, whereas we provide an efficient, tunable implementation demonstrated for ring, line, tree, mesh, and clique. Last but not least, our experiments demonstrate that PraSLE can easily scale up to networks of up to 80 nodes, which is an important property for practical use in IoT and CPS.

7 Conclusion and Future Work

We presented a practical self-stabilizing leader election algorithm, called PraSLE, supported by experimental validation on the IoT-Lab platform. We provided two versions of PraSLE, one for reliable networks and

another for unreliable networks. We proved the correctness of PraSLE and showed that it converges to a unique leader with probability 1. We also experimentally demonstrated that PraSLE outperforms existing methods in terms of average convergence time for ring, line, mesh, and clique topologies of up to 40 or 80 nodes. Our current experimental results are promising and point to the direction of sub-linear convergence time for non-linear topologies (which follows the asymptotic time complexity of PraSLE). However, we believe that we need to conduct more experiments with larger network sizes in order to verify this conjecture. The tunable nature of PraSLE enables us to optimize average convergence time, as well as communication and energy costs. Thus, PraSLE provides a resource-efficient primitive for distributed computing in unreliable networks of low-end IoT devices.

As an extension of this work, we are developing a variant of PraSLE where we elect the top k leaders instead of just one unique leader. Additionally, we plan to develop and evaluate other distributed algorithms (such as Paxos) on IoT platforms. We would also like to expand the scope of our evaluation to networks greater than 100 physical nodes and thoroughly evaluate more topologies such as grid.

Acknowledgment. We would like to thank David Hallberg for his help in the initial phases of this work, including algorithm brainstorming and scripting regarding energy consumption data. We'd also like to thank the IoT-Lab team for their publicly accessible large-scale testbed, which made our experimental evaluations possible.

References

[1] E. Baccelli, C. Gündoğan, O. Hahm, P. Kietzmann, M. S. Lenders, H. Petersen, K. Schleiser, T. C. Schmidt, and M. Wählisch. Riot: An open source operating system for low-end embedded devices in the iot. *IEEE Internet of Things Journal*, 5(6):4428–4440, 2018.

[2] A. Bounceur, M. Bezoui, R. Euler, N. Kadjouh, and F. Lalem. Brogo: A new low energy consumption algorithm for leader election in wsns. In *2017 10th International Conference on Developments in eSystems Engineering (DeSE)*, pages 218–223. IEEE, 2017.

[3] A. Bounceur, M. Bezoui, R. Euler, and F. Lalem. A wait-before-starting algorithm for fast, fault-tolerant and low energy leader election in wsns dedicated to smart-cities and iot. In *2017 IEEE SENSORS*, pages 1–3. IEEE, 2017.

[4] A. Bounceur, M. Bezoui, R. Euler, F. Lalem, and M. Lounis. A revised brogo algorithm for leader election in wireless sensor and iot networks. In *2017 IEEE SENSORS*, pages 1–3. IEEE, 2017.

[5] A. Bounceur, M. Bezoui, L. Lagadec, R. Euler, L. Abdelkader, and M. Hammoudeh. Dotro: A new dominating tree routing algorithm for efficient and fault-tolerant leader election in wsns and iot networks. In *International Conference on Mobile, Secure, and Programmable Networking*, pages 42–53. Springer, 2018.

[6] A. Bounceur, M. Bezoui, M. Lounis, R. Euler, and C. Teodorov. A new dominating tree routing algorithm for efficient leader election in iot networks. In *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–2. IEEE, 2018.

[7] A. Casteigts, Y. Métivier, J. M. Robson, and A. Zemmari. Deterministic leader election takes $\theta(d+\log n)$ bit rounds. *Algorithmica*, 81(5):1901–1920, 2019.

[8] Y.-J. Chang, T. Kopelowitz, S. Pettie, R. Wang, and W. Zhan. Exponential separations in the energy complexity of leader election. *ACM Transactions on Algorithms (TALG)*, 15(4):1–31, 2019.

[9] Q. Dong and D. Liu. Resilient cluster leader election for wireless sensor networks. In *2009 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 1–9. IEEE, 2009.

[10] D. Doty and D. Soloveichik. Stable leader election in population protocols requires linear time. *Distributed Computing*, 31(4):257–271, 2018.

[11] T. Faika, T. Kim, and M. Khan. An internet of things (iot)-based network for dispersed and decentralized wireless battery management systems. In *2018 IEEE Transportation Electrification Conference and Expo (ITEC)*, pages 1060–1064. IEEE, 2018.

[12] H. Garcia-Molina. Elections in a distributed computing system. *IEEE transactions on Computers*, (1):48–59, 1982.

[13] T. Herman. Probabilistic self-stabilization. *Information Processing Letters*, 35(2):63–67, 1990.

[14] N. Kadjouh, A. Bounceur, M. Bezoui, M. E. Khanouche, R. Euler, M. Hammoudeh, L. Lagadec, S. Jabbar, and F. Al-Turjman. A dominating tree based leader election algorithm for smart cities iot infrastructure. *Mobile Networks and Applications*, pages 1–14, 2020.

[15] K. Kothapalli, M. Onus, C. Scheideler, and C. Schindelhauer. Distributed coloring in $O(\sqrt{\log n})$ bits. In *Proc. of IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[16] L. Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[17] S. Lee, R. M. Muhammad, and C. Kim. A leader election algorithm within candidates on ad hoc mobile networks. In *International Conference on Embedded Software and Systems*, pages 728–738. Springer, 2007.

[18] M. Méndez, F. G. Tinetti, A. M. Duran, D. A. Obon, and N. G. Bartolome. Distributed algorithms on iot devices: bully leader election. In *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1351–1355. IEEE, 2017.

[19] M. U. Rahman. Leader election in the internet of things: Challenges and opportunities. *arXiv preprint arXiv:1911.00759*, 2019.

[20] A. Richa, C. Scheideler, S. Schmid, and J. Zhang. Self-stabilizing leader election for single-hop wireless networks despite jamming. In *Proceedings of the Twelfth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 1–10, 2011.

[21] N. Santoro. *Design and analysis of distributed algorithms*, volume 56. John Wiley & Sons, 2006.

[22] S. Sharma and A. K. Singh. An election algorithm to ensure the high availability of leader in large mobile ad hoc networks. *International Journal of Parallel, Emergent and Distributed Systems*, 33(2):172–196, 2018.

[23] A. V. Sheshashayee and S. Basagni. Wile: Leader election in wireless networks. *Ad Hoc Sens. Wirel. Networks*, 44(1-2):59–81, 2019.

[24] B. Sidik, R. Puzis, P. Zilberman, and Y. Elovici. Pale: Time bounded practical agile leader election. *IEEE Transactions on Parallel and Distributed Systems*, 31(2):470–485, 2019.

[25] S. Srinivasan and R. kandukoori. A synod based deterministic and indulgent leader election protocol for asynchronous large groups. *International Journal of Parallel, Emergent and Distributed Systems*, pages 1–28, 2021.

[26] Y. Sudo, F. Ooshita, T. Izumi, H. Kakugawa, and T. Masuzawa. Time-optimal leader election in population protocols. *IEEE Transactions on Parallel and Distributed Systems*, 2020.

[27] S. Vasudevan, J. Kurose, and D. Towsley. Design and analysis of a leader election algorithm for mobile ad hoc networks. In *Proceedings of the 12th IEEE International Conference on Network Protocols, 2004. ICNP 2004.*, pages 350–360. IEEE, 2004.

[28] R. Zhang, B. Jacquemot, K. Bakirci, S. Bartholme, K. Kaempf, B. Freydt, L. Montandon, S. Zhang, and O. Tonguz. Leader selection in vehicular ad-hoc network: a proactive approach. *arXiv preprint arXiv:1912.06776*, 2019.