Centrality-Based Eventual Leader Election in Dynamic Networks

Arnaud Favier, Luciana Arantes, Jonathan Lejeune and Pierre Sens
Inria, LIP6, CNRS, Sorbonne University, Paris, France
Email: {firstname.lastname}@lip6.fr

***Abstract*—This paper presents CEL, a new distributed eventual leader election algorithm for dynamic networks, which exploits topological information to improve the choice of a central leader and reduce message exchanges. The algorithm has a cross-layer neighbors detection, with a neighbor-aware mechanism, to improve the sharing of topological knowledge and elect a central leader faster. It uses a self-pruning mechanism based on topological knowledge, combined with probabilistic gossip, to improve the performance of broadcast propagation. Evaluations were conducted on the OMNeT++ environment, simulating realistic MANET with interference, collision, and messages loss. Using different parameters values, we have compared CEL to Gómez-Calzado *et al.* algorithm [1], on the Random Walk and the Truncated Lévy Walk mobility models. The results show better performances than [1], including fewer messages sent, shortest paths to the leader, and a more stable algorithm.**

***Index Terms*—distributed systems, dynamic networks, MANET, leader election**

I. Introduction

Leader election is a key component for many fault-tolerant services in asynchronous distributed systems. By coordinating actions of a set of distributed processes, also called nodes, it eases to solve agreement problems, like the consensus. Consensus is a fundamental problem of distributed computing [2], used by many other problems in the literature, such as state machine replication or atomic broadcast. Several consensus algorithms such as Paxos [3] and many blockchains like Bitcoin [4] adopt a leader-based approach. They rely on an eventual leader election service, also known as the Ω failure detector [5]. Ω provides a primitive called *Leader()*, which, when invoked, returns the identity of a process in the system and guarantees that there is a time after which it always returns the identity of the same correct process.

Many leadership protocols were proposed in the literature to implement Ω. Some of them consider static distributed systems [2], [6]–[8] and rely on a model, where the membership of the system is known in advance and the topology of the underlying network does not change. Among the ones considering dynamic issues [9]–[15], only a few take into account the characteristics and the lack of knowledge of highly dynamic systems. Furthermore, most of these algorithms do not choose the leader according to a topological criterion, i.e., the position of the leader in the network. The topological position of the leader has a strong impact on the performance of algorithms using the leader election service, since the leader must collect information from the other processes, such as from a majority of processes in the case of consensus. Thus, the average number of hops to reach the leader has a direct impact on the performance of consensus algorithms.

This paper proposes a new eventual leader election algorithm for dynamic systems. We assume that nodes can move and communicate by sending messages over wireless links, and that system membership is not known in advance. The communication graph can evolve over time, therefore, the network is not always fully connected but composed of one or more connected components. Our algorithm chooses the leader according to a topological criterion: for every component, the leader is eventually the node having the best closeness centrality in the component. Each node progressively builds and maintains a local knowledge of the component communication graph. This knowledge is then used to locally determine a central leader, well located to be reached by a majority of processes.

The current paper brings three main contributions:

1) A new Centrality-based Eventual Leader election algorithm for dynamic systems, called *CEL*, where the leader eventually has the best centrality. *CEL* has a cross-layer neighbors detection which exploits the broadcast features of the underlying wireless network. The neighbor-aware mechanism improves the sharing of the topological knowledge and elects a central leader faster.

2) CEL uses the topological knowledge through a self-pruning mechanism, combined with probabilistic gossip, to reduce global information propagation costs.

3) An extensive evaluation on the OMNeT++ environment [16] using two mobility patterns to simulate mobile ad hoc networks (MANET) with interference, collision, and messages loss. Comparison with the closest algorithm [1] to our work shows that *CEL* has a good trade-off considering the number of messages exchanged, stability of the leader (i.e., the percentage of the average time that nodes adopt the expected leader), and the closeness of the leader to the other nodes.

The rest of the paper is organized as follows: Section II presents related research works to the leader election problem, Section III explains the system model and assumptions, Section IV describes the algorithm, Section V discusses performance results, and finally, a conclusion and future work are given in Section VI.

II. Related work

Several protocols were proposed in the literature to implement Ω in asynchronous systems prone to crash failures, taking into account a static model [17]–[19]. Since it is impossible to solve an eventual leader election in such systems [20], additional assumptions are necessary. The majority of works use one of the two orthogonal approaches: timer-based, which assumes that links are eventually timely [17], [19], or message exchange pattern-based [21], i.e., a query-response mechanism where eventually there is a link whose responses are always received before the others. A punishment mechanism when nodes do not send their message on time is then used as a criterion to elect the most stable leader.

Leader election in a dynamic context has been studied by several authors. In [9], Malpani *et al.* build an acyclic graph on top of a dynamic topology where each node has a direct path to the leader. However, the choice of the leader is based on a movement-based counter and does not take into account the underlying network topology. This algorithm was extended by [12] and [13] where the election of a new leader requires three diffusion waves over the network. Such waves induce a high number of messages, which slow down the eventual election of the leader.

Rahman *et al.* use in [11] the highest identifier node as the criterion to elect a node. The algorithm builds a spanning tree and requires heartbeat, probe, reply, and acknowledgment messages. Therefore, the number of exchanged messages is very high, and can overload the network. Vasudevan *et al.* [10] use a wave algorithm to build a spanning tree, with a static election criterion based on the highest arbitrary value initially given to a node. However, a leader is elected only when it is accepted by all nodes in the network. The algorithm proposed by Kim *et al.* [15] also builds a spanning tree to elect a centrally positioned leader, according to the average depth of nodes in the tree. However the central leader is not always optimal, depending on the initiator node of the election, and the mobility of nodes is not studied.

The leader election algorithm for dynamic network proposed in [14] uses a message exchange pattern-based. Authors consider some eventual network stability assumptions, and the election criterion is only based on a punishment approach regardless of the topological position of the leader.

In our previous work [22], the algorithm elects a central leader, but assumes reliable communication channels, which are not suitable for realistic environments with message interference and collisions. A global view of the network is exchanged using probes and an update mechanism, leading to message collisions and losses. Furthermore, the knowledge of the topology is not used to improve communication performance, and the bidirectional links assumption is not taken into account to optimize knowledge sharing.

Gómez-Calzado *et al.* use in [1] a mechanism to detect whether a node is connected to other nodes. Periodically, unconnected nodes send *join* messages, while among the connected nodes, only the leader initiates the sending of messages, which are then relayed by other nodes. Such an approach induces a large number of messages to be sent. The elected leader will be the oldest node of the connected component with the highest identifier. Therefore, the leader is not necessarily the best-located node to be reached by a majority of nodes.

III. System model and assumptions

The system is considered partially synchronous, with two unknown upper bounds: δ on the transmission delay (eventual timely links), and ϕ on the time taken by a node to execute a step. We consider one process per node, therefore, the words *node* and *process* are interchangeable.

Node states and failures: Nodes always follow the specification of the algorithm until they fail. They can fail by crashing and a node can recover, joining the system again with the same *unique identifier* as before the failure. Hence, a node keeps its identifier regardless of its state, and two nodes cannot have the same *identifier*. However, a node does not recover its state neither its knowledge of the network membership, thus, is initialized again.

Initially, all nodes in the system are in the *correct* state. A node is considered *faulty* if it fails and does not recover, or if it leaves the system forever. Otherwise, if present in the system, it is considered *correct*.

Communication graph: The system is modeled as an undirected graph, where a vertex corresponds to a node, and an edge represents a communication link between two nodes, i.e., a distance of 1-hop. Two nodes can communicate directly if they are in the transmission range of each other, i.e., a receiver node is located inside the emission range of a sender node. In our system, the emission range is the same as the reception range. Therefore, if node i can communicate with node j, node j can also communicate with node i (bidirectional links).

Adjacent vertices of a node are called *neighbors*, and the set of them is called the *neighborhood* of this node. A given node belongs to a connected graph formed by its neighbors, neighbors of its neighbors, and so on, which we called a *connected component*.

Due to the movement, failure, and disconnection of nodes, the system can be divided into two (or more) different connected components. Each of them is considered to be a fully-fledged network in itself, and therefore, eventually elects one leader. Both partial synchrony and algorithm ensure that, regardless of topology changes, if the changes cease, each connected component will eventually elect a single leader.

Channels: Nodes can only communicate by broadcasting local messages, which are received by all neighbors of the sending node. Communication is based on a fixed Wi-Fi channel, chosen beforehand. We consider *unreliable* communication channels with messages losses, inducted by messages interference and collisions. Therefore, the CSMA/CA protocol included in IEEE 802.11 [23], is used to handle messages losses. There are no assumptions about message ordering, i.e., messages can be delivered out of order.

Algorithm 1: *Centrality-based Eventual Leader (CEL)* election algorithm for node i

```
1  Typedef view: ⟨clock: int, neigh: set(id)⟩
2  Message knowledge: ⟨map(key: id, value: view)⟩
3  Local variables:
4      known: map(key: id, value: view)
5  Initialization:
6      known[i].neigh ← {i}
7      known[i].clock ← 0
8  Connection of node j:
9      known[i].neigh ← known[i].neigh ∪ {j}
10     known[i].clock ← known[i].clock + 1
11     if j ∉ known then
12         known[j].neigh ← {j, i}
13         known[j].clock ← 1
14     else
15         known[j].neigh ← known[j].neigh ∪ {i}
16         known[j].clock ← known[j].clock + 1
17     LocalBroadcast (knowledge⟨known⟩, 1)
18 Disconnection of node j:
19     known[i].neigh ← known[i].neigh \ {j}
20     known[i].clock ← known[i].clock + 1
21     known[j].neigh ← known[j].neigh \ {i}
22     known[j].clock ← known[j].clock + 1
23     LocalBroadcast (knowledge⟨known⟩, 1)
24 Receive knowledge message known_j from node j:
25     ∀ n ∈ known_j do
26         if n ∉ known or
27         known_j[n].clock > known[n].clock then
28             known[n] ← known_j[n]
29             UpdateNeighbors (known_j, n)
30         else if known_j[n].clock = known[n].clock then
31             known[n].neigh ← known[n].neigh ∪
                known_j[n].neigh
32             UpdateNeighbors (known_j, n)
33     if known was updated then
34         TopologicalBroadcast ()
35 Call of UpdateNeighbors(known_j, n):
36     ∀ k ∈ known_j[n].neigh do
37         if k ∉ known or
38         known_j[k].clock > known[k].clock then
39             known[k] ← known_j[k]
40         else if known_j[k].clock = known[k].clock then
41             known[k].neigh ← known[k].neigh ∪
                known_j[k].neigh
42 Call of TopologicalBroadcast():
43     ∀ n ∈ known[i].neigh do
44         if known[n].neigh = known[i].neigh then
45             if n < i then
46                 return
47     LocalBroadcast (knowledge⟨known⟩, ρ)
48 Invocation of Leader():
49     component ← known[i].neigh
50     ∀ j ∈ component do
51         component ∪ known[j].neigh
52     return Max (ClosenessCentrality (component))
```

Membership and nodes identity: Initially, each node only knows its unique *identifier* in the system. This means that nodes do not know the total number of nodes, neither the membership of the system. Nodes detect their neighbors through a *cross-layer* mechanism described in Section IV-B, using already existing *beacon* messages of the data link layer.

IV. Leader Election Algorithm

This section presents the *Centrality-based Eventual Leader (CEL)* election algorithm. The pseudo-code for node i is given in Algorithm 1. In *CEL*, every node maintains a topological knowledge of the connected component to which it belongs. The algorithm builds this knowledge during node connections and disconnections (triggered by the *cross-layer* mechanism), and by sending *knowledge* messages to its neighbors. Nodes spread *knowledge* messages using *probabilistic gossip*, combined with a *self-pruning* mechanism that exploits the topological knowledge to reduce the number of messages sent. A new *knowledge* message is only sent after a connection or disconnection. Based on the component knowledge, the algorithm eventually elects one leader per component, which is placed at the center of the component.

A. Data structures, messages, and variables

CEL uses a data structure called a **view** (line 1). A *view* associated to node i is composed of two elements: 1) a logical *clock* value, acting as a timestamp and incremented at each connection and disconnection; 2) a set of node *identifiers*, which are the current neighbors of i.

Each node i maintains a local variable (line 3) called **known**. This variable represents the current topological knowledge that i has of its current component (including itself).

The only type of message exchanged between neighbors is the **knowledge** message (line 2). It contains the current topological knowledge that the sender node has of the network, i.e., its *known* variable.

B. Description of the algorithm

Firstly, node i initializes its *known* variable with its own identifier (i), and sets its logical clock to 0 (lines 5 to 7).

a) Node connection: When a new node j appears in the transmission range of i, the cross-layer mechanism of i detects j, and triggers the *Connection* method (line 8). Node j is added to the neighbors set of node i (line 9). As the knowledge of i has been updated, its logical clock is incremented (line 10). Since links are assumed bidirectional, i.e., the emission range equals the reception range, if node i has no previous knowledge of j (line 11), the neighbor-aware

mechanism adds both i and j in the set of neighbors of j (line 12). Then, i sets the clock value of j to 1 (line 13), as i was added to the knowledge of node j. On the other hand, if node i already has information about j (line 14), i is added to the neighbors of j (line 15), and the logical clock of node j is incremented (line 16). Finally, by calling *LocalBroadcast* method, node i shares its knowledge with j and informs its neighborhood of its new neighbor j. Note that such a method sends a knowledge message to the neighbors of node i, with a gossip probability ρ [24]. However, for the first hop, ρ is set to 1 to make sure that all neighbors of i will be aware of its new neighbor j. Note that the cross-layer mechanism of node j will also trigger its *Connection* method, and the respective steps will also be achieved on node j.

b) Node disconnection: When a node j disappears from the transmission range of node i, the cross-layer mechanism stops receiving beacon messages at the MAC level, and triggers the *Disconnection* method (line 18). Node j is then removed from the knowledge of node i (line 19), and its clock is incremented as its knowledge was modified (line 20). Then, the neighbor-aware mechanism assumes that node i will also disconnect from j. Therefore, i is removed from the neighborhood of j in the knowledge of node i, and the corresponding clock is incremented (lines 21- 22). Finally, node i broadcasts its updated knowledge to its neighbors (line 23).

c) Knowledge update: When node i receives a knowledge message $known_j$, from node j (line 24), it looks at each node n included in $known_j$ (line 25). If n is an unknown node for i (line 26), or if n is known by node i and has a more recent clock value in $known_j$ (line 27), the clock and neighbors of node n are updated in the knowledge of i (line 28). Note that a clock value of n higher than the one currently known by node i (line 27) means that node n made some connections and/or disconnections of which node i is not aware. Then, the *UpdateNeighbors* method is called to update the knowledge of i regarding the neighbors of n (line 29). If the clock value of node n is identical to the one of both the knowledge of node i and $known_j$ (line 30), the neighbor-aware mechanism merges the neighbors of node n from $known_j$ with the known neighbors of n in the knowledge of i (line 31). Remark that the clock of node n is not updated by the neighbor-aware mechanism, otherwise, n would not be able to override this view in the future with more recent information. The *UpdateNeighbors* method is then called (line 32). Finally, node i broadcasts its knowledge only if this latter was modified (lines 33-34).

The *UpdateNeighbors* method (line 35) updates the knowledge of i with information about the neighbors of node n (line 36). If the neighbor k is an unknown node for i (line 37), or if k is known by i but has a more recent clock value in $known_j$ (line 38), the clock and neighbors of node k are added or updated in the knowledge of node i (line 39). Otherwise, if the clock of node k is identical in the knowledge of node i and in $known_j$ (line 40), the neighbor-aware mechanism merges the neighbors of node k in the knowledge of i (line 41).

d) Information propagation: The *TopologicalBroadcast* method (line 42) uses a self-pruning approach [25] to broadcast or not the updated knowledge of node i, after the reception of a knowledge from a neighbor j. To this end, node i checks whether each of its neighbors has the same neighborhood as itself (lines 43 to 44). In this case, node n is supposed to have also received the knowledge message from neighbor node j. Therefore, among the neighbors having the same neighborhood than i, only the one with the smallest identifier will broadcast the knowledge (line 45), with a gossip probability ρ (line 47).

e) Leader election: The leader is elected when a process running on node i calls the *Leader* function (line 48). This function returns the most central leader in the component according the closeness centrality (line 52), using the knowledge of node i. The closeness centrality characterizes the ability of a node to spread information over the network. For a node x, the closeness centrality is the inverse of the sum of all shortest paths to other nodes, defined by Alex Bavelas [26] with the following formula:

$$C(x) = \frac{1}{\sum_y d(y,x)}$$

where $d(y,x)$ is the shortest path between nodes y and x. We chose the closeness centrality instead of the betweenness centrality, because it is faster to compute and requires fewer computational steps, therefore consuming less energy from the mobile node batteries than the latter.

First, node i rebuilds its component according to its topological knowledge. To do so, it computes the entire set of reachable nodes, by adding neighbors, neighbors of its neighbors, and so on (lines 49 to 51). Then, it evaluates the shortest distance between each reachable node and the other ones, and computes the closeness centrality for each of them. Finally, it returns the node having the highest closeness value as the leader (line 52). The highest node identifier is used to break ties among identical centrality values. If all nodes of the component have the same topological knowledge, the *Leader()* function will return the same leader node when invoked. Otherwise, it may return different leader nodes. However, when the network topology stops changing, our algorithm ensures that all nodes of a component will eventually have the same topological knowledge and therefore, the *Leader()* function will return the same leader node for all of them [13].

V. Results

Realistic simulations were carried over in order to compare our *Centrality-based Eventual Leader* (*CEL*) algorithm, with the Ω eventual leader election algorithm of Gómez-Calzado *et al.* (see Section II).

A. Simulation environment

Experiments were conducted on a C++ discrete event simulator called OMNeT++ [27], with the INET framework [28] to model wireless protocols and mobile networks. This environment allows simulation of unreliable communication channels

and realistic layers of the OSI communication model. Each experiment involves 60 moving nodes placed in a 500 $\times$ 500 meters obstacle-free area during 30 simulated minutes.

Simulations consider a full MANET network stack, with the physical and data-link layers following the IEEE 802.11n specifications [23]. An additional use of a cross-layer mechanism at the MAC level, allows the application layer to access neighbor's MAC addresses. Therefore, the identifier of a node is a MAC address, encoded on 3 bytes rather than usually 6 bytes, as we assume that all nodes have network components from the same manufacturer. Every node uses a single transceiver, with a fixed transmission range decided at the beginning of the experiment between 20m and 80m, and identical for all nodes. This transceiver uses the 2.4 GHz frequency band, with a nominal bitrate of 52 Mbps.

B. Mobility models

Experiments use two mobility models from BonnMotion [29], a mobility scenario generation tool: (1) *Random Walk* [30], where a node moves from its current location to a new location by randomly choosing a direction in the interval [0, 2π] and a speed between 0.1m/s and 1m/s, with a pause time of 10 seconds once the destination is reached; (2) *Truncated Lévy Walk* [31], which characterizes human mobility. Lévy walks are continuous-time random walks whose turning points are the visit points associated with the Lévy flights model. Parameters are a flight length coefficient α sets to 1 and a pause time coefficient β sets to 1.

C. Algorithms settings

In *CEL*, beacon messages are sent every $\tau = 102.4$ milliseconds (usual interval value of the Target Beacon Transmission Time), detected at the MAC level by the cross-layer mechanism. We evaluated two configurations of the algorithm:

- In *CEL-1*, the probability ρ to gossip a knowledge message in the *TopologicalBroadcast* method is set to 1. Therefore, messages are flooded in the network, if the neighborhood of the sender node is different from the receiver's one.
- In *CEL-0.7*, ρ is set to 0.7, i.e., a message is retransmitted with 0.7 probability [24], if the sender and receiver neighborhoods are different.

In Gómez-Calzado *et al.* algorithm, the frequency to send either $join$ messages when a node is unconnected, or leader messages, is 102.4ms, as both are considered beacon messages. The timers detecting leader failure and node disconnection are initialized to 100ms and increased by 500ms when they expire. We ran multiple experiments and found these values were the best.

D. Metrics

We considered the following three metrics:

1) Average number of messages sent per second per node: This metric does not consider *beacon* messages, since the same number of *beacons* is sent every τ milliseconds in both algorithms. In *CEL*, *beacon* messages are sent by the underlying MAC layer.

2) Average of the median path to the leader: This metric characterizes how fast a leader can be reached a majority of nodes (at least 50%) in its component. We compute the *longest path* of all shortest paths from every node to their current leader, except for single node components, as its null path would unfairly improve the metric. Then, we compute the average of all medians over time.

3) Instability: The percentage of the average time a node elects a different leader from the eventual elected one. The latter is deduced by an oracle, based on the closeness centrality for *CEL*, and on the oldest node of the connected component with the highest identifier for Gómez-Calzado *et al.* algorithm. First, we compute the $CurrentInstability$ at time t with the following formula:

$$CurrentInstability_t = \frac{\sum_{i=0}^{N} \begin{cases} 0 \text{ if } leader_t(i) = oracle_t(i) \\ 1 \text{ if } leader_t(i) \neq oracle_t(i) \end{cases}}{N}$$

where N is the number of nodes in the system, and i the node identifier. Then, we compute the *Instability* over the entire experiment time, which is the average $CurrentInstability_t$ from 0, to the end of the experiment (1 800 seconds).

E. Performance results

The goal is to compare the performance of both versions of the *CEL* algorithm, with Gómez-Calzado algorithm [1], using different transmission ranges on both mobility patterns. Note that the number of components in the system is strongly correlated with the transmission range.

1) Average number of messages sent per second is shown in Figures 1 and 2 for both mobility patterns. Right y-axes give the average number of components and their average diameter. In the Gómez-Calzado algorithm, nodes periodically send *join* messages when they are alone, in order to connect with a bigger component.

On both mobility models, *CEL-1* sends more messages than *CEL-0.7*, as it floods the network by broadcasting every received knowledge message. *CEL-0.7* reduces the number of messages sent per second, especially in larger transmission ranges, where more messages are broadcast at each topological change. There is an average reduction of 36% when ρ is set to 0.7 compared to ρ sets to 1, on both mobility patterns for a transmission range from 20m to 80m.
Note that the average message sent size varies from 4.56 to 6.58 bytes in the Gómez-Calzado algorithm, and from 263.03 to 1322.69 bytes for both versions of the *CEL* algorithm, as they share the topological knowledge of the component, and fits into the MTU of a single network packet.

2) Average of the median path to the leader is shown in Figures 3 and 4 for both mobility patterns. In Figure 3, we observe that the average median path from all nodes of the components to the leader, is shorter in *CEL* than in Gómez-Calzado algorithm for the Random Walk mobility model, with a gain of up to 26% in larger transmission ranges. Interestingly, the probabilistic gossip version of *CEL* has a low impact on the leader path.

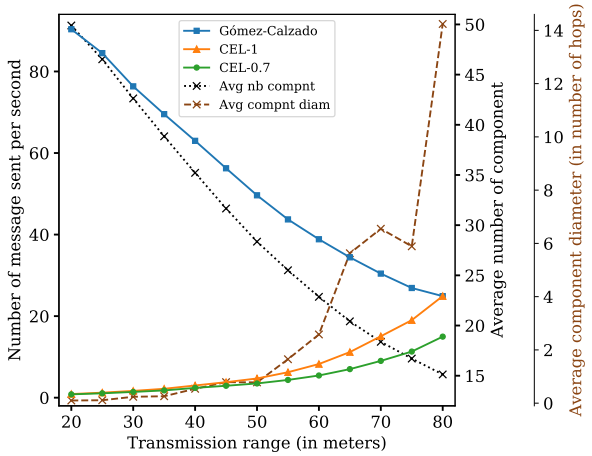

Figure 1: Messages sent (lower is better) Random Walk

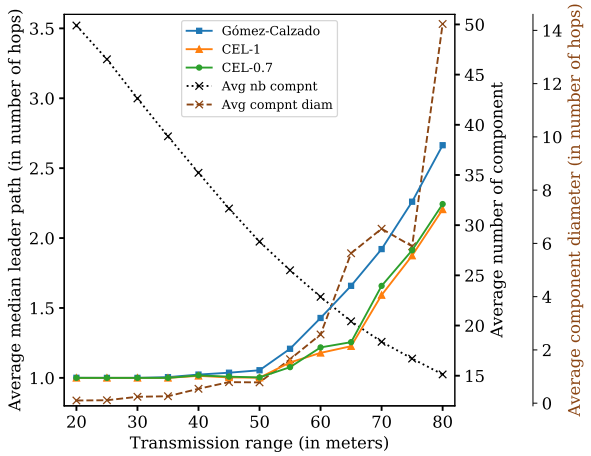

Figure 3: Leader path (lower is better) Random Walk

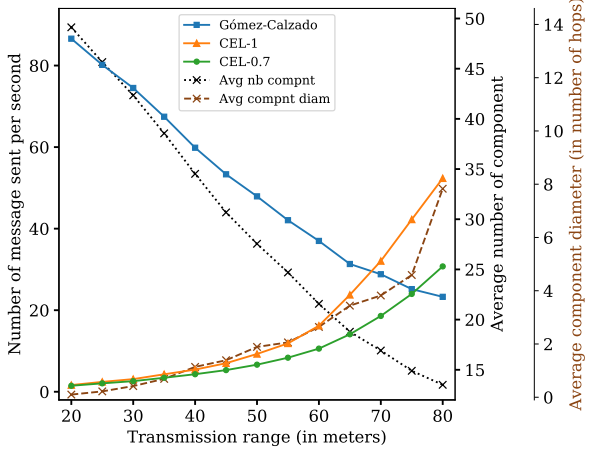

Figure 2: Messages sent (lower is better) Truncated Lévy Walk

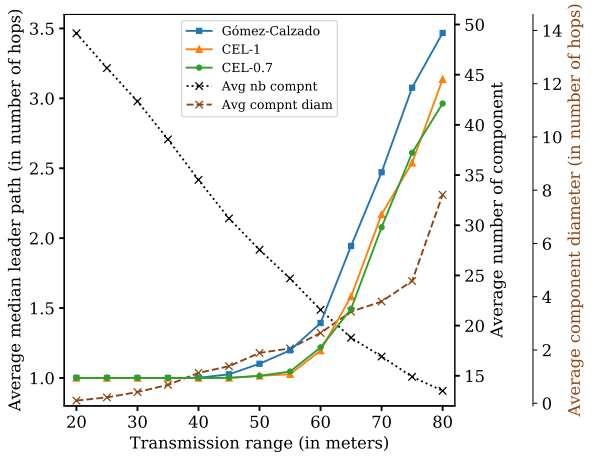

Figure 4: Leader path (lower is better) Truncated Lévy Walk

The Truncated Lévy Walk pattern shows in Figure 4 the impact of flying nodes which disrupt the component, by quickly moving in and out, therefore, modifying potential paths to the leader. Therefore, the difference between the *CEL* algorithm with ρ sets to 0.7 and the Gómez-Calzado algorithm, leads to a shortest path up to 15% for the probabilistic gossip version of *CEL*.

Sharing a topological knowledge like in the *CEL* algorithm, allows the election of a central leader per component. Consequently, the results confirm that the number of hops to reach the leader by the nodes of its component is reduced.

3) Instability evolution is shown in Figures 5 and 6, according to the transmission range, and for both patterns. First, we observe that the average instability increases when the transmission range increases, since components are composed of more nodes with a larger diameter. Hence, it takes a longer time to elect a new leader for all the algorithms.

In Figure 5, the percentage of instability in Gómez-Calzado algorithm is on average 69% higher than on both *CEL* versions. There is no significant instability difference between the *CEL* versions. The instability for the Truncated Lévy Walk pattern in Figure 6, shows that the *CEL* algorithm is more stable than Gómez-Calzado algorithm when the transmission range increases. On average, *CEL* versions are 57% more stable than Gómez-Calzado algorithm. We also observe that the probabilistic gossip version of *CEL* with ρ sets to 0.7, is slightly less stable than the flooding version with ρ sets to 1. This is induced by a lower number of broadcast messages, making disrupting changes caused by flying nodes, to take more time to be spread over large components diameter.

4) Focusing on the 60 meters range over time is interesting to understand in detail the differences between the algorithms behaviors, on an approximate range of usual Wi-Fi indoor devices. Figures 7 and 8 show the average instability from time 0 to time t for both mobility patterns. The right y-axis gives the exact number of components at time t.

In Figure 7, at the beginning of the experiment on the Random Walk pattern, Gómez-Calzado algorithm has a higher instability rate, which quickly decreases to reach a threshold of 50% at 240 seconds, with a slight increase over time. Both *CEL* algorithm versions need a few seconds to stabilize, before reaching a threshold of around 430 seconds. The probabilistic gossip version of *CEL* is less stable than the flooding version,

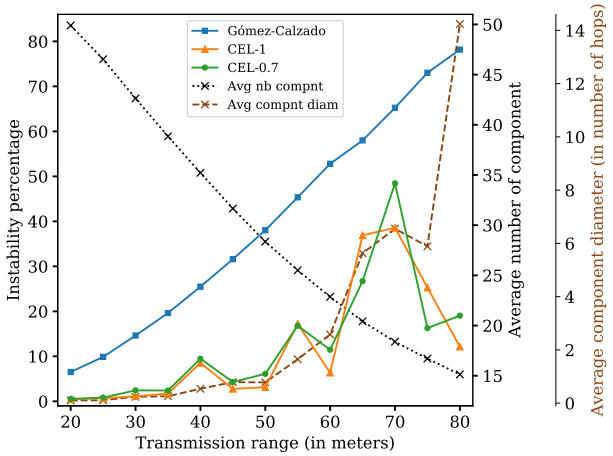

Figure 5: Instability (lower is better) Random Walk

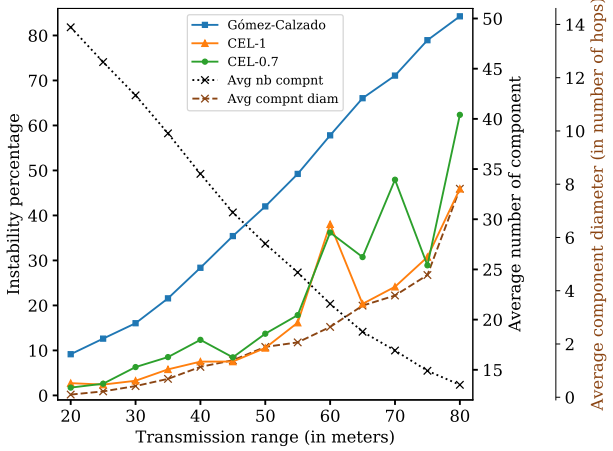

Figure 6: Instability (lower is better) Truncated Lévy Walk

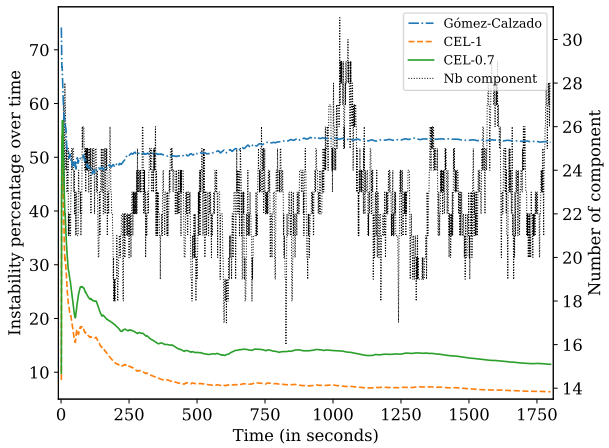

Figure 7: Instability at 60m (lower is better) Random Walk

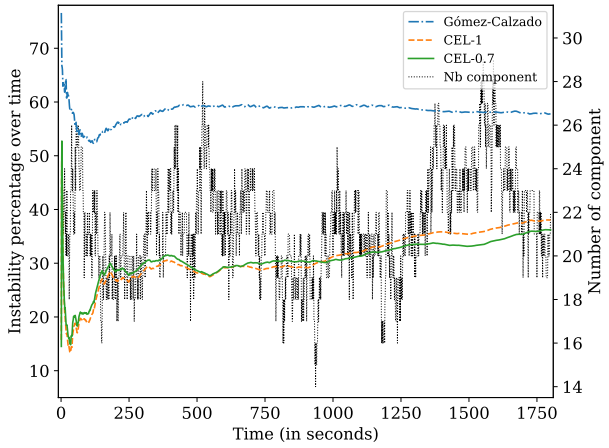

Figure 8: Instability at 60m (lower is better) Truncated Lévy Walk

as some knowledge messages are not broadcast by nodes to their neighborhood.

Figure 8 shows that the Truncated Lévy Walk model increases the instability of Gómez-Calzado algorithm, where an instability threshold of 59% is reached after 416 seconds. On the other hand, the *CEL* versions have a common instability evolution over time, with a small difference at the end of the experiment following the rebroadcast probability.

5) A comparative analysis with Topology Aware [22] presented in Section II, shows the performance gain by both the neighbor-aware and self-pruning mechanisms which exploit the topological knowledge, and by the probabilistic gossip. Experiments ran in the realistic OMNeT++ environment described in Section V-A, with a probe frequency $\tau = 102.4$ms for Topology Aware. The analysis is focused on a WiFi transmission range of 80 meters, as it is the highest range of our experiments and a complex configuration with large components diameters. Results for the Random Walk mobility pattern in Table I, show that exploiting the topological knowledge reduces the number of messages sent per second by 71.92% comparing the Topology Aware algorithm to *CEL-0.7*, while having a shorter leader path. Instability is 44.14% lower comparing Topology Aware to *CEL-1*. For the Truncated Levy Walk pattern in Table II, the comparison between Topology Aware and *CEL-0.7* shows a reduction of the number of messages sent per second by 63.43%, and an average median leader path lower by 15.43%. Note that while the instability percentage is lower for *CEL-1*, the reduction of the number of messages by the probabilistic gossip version may lead to lower stability than Topology Aware, especially when high transmission ranges imply large components.

	Topology Aware	CEL-1	CEL-0.7
Messages sent	53.32/s	24.91/s	14.97/s
Leader path (in hop)	2.44	2.20	2.24
Instability	21.75%	12.15%	19.04%

Table I: Random Walk (lower is better)

	Topology Aware	CEL-1	CEL-0.7
Messages sent	84.06/s	52.35/s	30.74/s
Leader path (in hop)	3.50	3.14	2.96
Instability	54.53%	45.92%	62.36%

Table II: Truncated Levy Walk (lower is better)

VI. Conclusion and future work

This paper proposed *CEL*, a new distributed eventual leader election algorithm for dynamic wireless networks, which exploits topological information to improve the choice of the leader and reduce message exchanges. A leader is eventually elected in each connected component of the network, and it is the node having the highest closeness centrality in the communication graph of the connected component. We argue that the leader choice is a key issue: for instance, agreement algorithms such as Paxos [3], often use a leader to collect node proposals. Therefore, the average number of hops to reach the leader has an impact on the performance of such algorithms. To compute the centrality and then elect the current leader, each node maintains the knowledge of the network topology. Initially, each node only knows itself and, by exchanging messages with neighbors, progressively builds its knowledge of the network topology.

Our algorithm adopts a cross-layer approach: when the underlying network layer (MAC) detects a change in the current neighborhood, the node updates its knowledge and spreads its new view of the network. In order to reduce the cost of message propagation, we adopt a probabilistic gossip approach and use local topological information to avoid redundant broadcasts. Evaluation results from experiments on the OMNet++ environment with two mobility models, *Random Walk* and *Truncated Lévy Walk*, confirm that our algorithm reduces the number of messages and the path to the leader, when compared to Gómez-Calzado *et al.* algorithm.

As future research directions, we plan to reduce the size of messages exchanged, by using compression algorithms, or by restricting the knowledge of the network to a limited number of hops. We also intend to estimate the energy consumption of nodes. Finally, we plan to explore the possibilities of collaborative computations for centralities.

References

[1] C. Gómez-Calzado, A. Lafuente, M. Larrea, and M. Raynal, "Fault-tolerant leader election in mobile dynamic distributed systems," in *19th Pacific Rim Int. Symposium on Dependable Computing*. IEEE, 2013, pp. 78–87.

[2] D. Peleg, "Time-optimal leader election in general networks," *Journal of parallel and distributed computing*, vol. 8, no. 1, pp. 96–99, 1990.

[3] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems (TOCS)*, vol. 16, no. 2, pp. 133–169, 1998.

[4] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Oct. 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[5] T. D. Chandra, V. Hadzilacos, and S. Toueg, "The weakest failure detector for solving consensus," *Journal of the ACM (JACM)*, vol. 43, no. 4, pp. 685–722, 1996.

[6] G. L. Peterson, "An o (n log n) unidirectional algorithm for the circular extrema problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 4, pp. 758–762, 1982.

[7] G. N. Frederickson and N. A. Lynch, "The impact of synchronous communication on the problem of electing a leader in a ring," in *The 16th annual ACM symposium on Theory of computing*, 1984, pp. 493–503.

[8] N. A. Lynch, *Distributed algorithms*. Elsevier, 1996, ch. 4.1, pp. 52–56.

[9] N. Malpani, J. L. Welch, and N. Vaidya, "Leader election algorithms for mobile ad hoc networks," in *The 4th int. workshop on Discrete algorithms and methods for mobile computing and communications*. ACM, 2000, pp. 96–103.

[10] S. Vasudevan, J. Kurose, and D. Towsley, "Design and analysis of a leader election algorithm for mobile ad hoc networks," in *The 12th int. Conference on Network Protocols, ICNP.* IEEE, 2004, pp. 350–360.

[11] M. Rahman, M. Abdullah-Al-Wadud, and O. Chae, "Performance analysis of leader election algorithms in mobile ad hoc networks," *Int. J. of Computer Science and Network Security*, vol. 8, no. 2, pp. 257–263, 2008.

[12] R. Ingram, P. Shields, J. E. Walter, and J. L. Welch, "An asynchronous leader election algorithm for dynamic networks," in *Int. Symposium on Parallel & Distributed Processing*. IEEE, 2009, pp. 1–12.

[13] R. Ingram, T. Radeva, P. Shields, S. Viqar, J. E. Walter, and J. L. Welch, "A leader election algorithm for dynamic networks with causal clocks," *Distributed computing*, vol. 26, no. 2, pp. 75–97, 2013.

[14] L. Arantes, F. Greve, P. Sens, and V. Simon, "Eventual leader election in evolving mobile networks," in *Int. Conference On Principles Of Distributed Systems*. Springer, 2013, pp. 23–37.

[15] C. Kim and M. Wu, "Leader election on tree-based centrality in ad hoc networks," *Telecommunication Systems*, vol. 52, no. 2, pp. 661–670, 2013.

[16] A. Varga, "Omnet++," in *Modeling and tools for network simulation*. Springer, 2010, pp. 35–59.

[17] M. Larrea, A. Fernández, and S. Arévalo, "Optimal implementation of the weakest failure detector for solving consensus," in *19th IEEE Symposium on Reliable Distributed Systems, SRDS'00, Proc.*, 2000, pp. 52–59.

[18] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, "Stable leader election," in *Distributed Computing, 15th Int. Conference, DISC 2001, Proc.*, 2001, pp. 108–122.

[19] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, "On implementing omega with weak reliability and synchrony assumptions," in *The 22nd annual symposium on Principles of distributed computing*, 2003, pp. 306–314.

[20] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, p. 374–382, Apr. 1985.

[21] A. Mostéfaoui, E. Mourgaya, M. Raynal, and C. Travers, "A time-free assumption to implement eventual leadership," *Parallel Process. Lett.*, vol. 16, no. 2, pp. 189–208, 2006.

[22] A. Favier, N. Guittonneau, L. Arantes, A. Fladenmuller, J. Lejeune, and P. Sens, "Topology aware leader election algorithm for dynamic networks," in *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 2020, pp. 1–10.

[23] "Ieee std 802.11n-2009 – part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 5: Enhancements for higher throughput," 2009.

[24] Z. J. Haas, J. Y. Halpern, and L. Li, "Gossip-based ad hoc routing," in *Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3. IEEE, 2002, pp. 1707–1716.

[25] H. Lim and C. Kim, "Flooding in wireless ad hoc networks," *Computer Communications*, vol. 24, no. 3-4, pp. 353–363, 2001.

[26] A. Bavelas, "Communication patterns in task-oriented groups," *The journal of the acoustical society of America*, vol. 22, no. 6, pp. 725–730, 1950.

[27] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, 2008, pp. 1–10.

[28] L. Mészáros, A. Varga, and M. Kirsche, "Inet framework," in *Recent Advances in Network Simulation*. Springer, 2019, pp. 55–106.

[29] N. Aschenbruck, R. Ernst, E. Gerhards-Padilla, and M. Schwamborn, "Bonnmotion: a mobility scenario generation and analysis tool," in *Proceedings of the 3rd international ICST conference on simulation tools and techniques*, 2010, pp. 1–10.

[30] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless communications and mobile computing*, vol. 2, no. 5, pp. 483–502, 2002.

[31] I. Rhee, M. Shin, S. Hong, K. Lee, S. J. Kim, and S. Chong, "On the levy-walk nature of human mobility," *IEEE/ACM transactions on networking*, vol. 19, no. 3, pp. 630–643, 2011.