Eventual Leader Election in Evolving Mobile Networks

Luciana Arantes[1], Fabíola Greve[2], Pierre Sens[1], and Véronique Simon[1]

[1] LIP6, Université Pierre et Marie Curie, Inria, CNRS, France
{firstname.lastname}@lip6.fr, Fax: +33-1-44-27-74-95
[2] DCC - Computer Science Department / Federal University of Bahia, Brazil
fabiola@dcc.ufba.br

Abstract. Many reliable distributed services rely on an eventual leader election to coordinate actions. The eventual leader detector has been proposed as a way to implement such an abstraction. It ensures that, eventually, each process in the system will be provided by an unique leader, elected among the set of correct processes in spite of crashes and uncertainties. A number of eventual leader election protocols were suggested. Nonetheless, as far as we are aware of, no one of these protocols tolerates a free pattern of node mobility. This paper proposes a new protocol for this scenario of dynamic and mobile unknown networks.

Keywords: Fault-tolerant leader election, dynamic networks, process mobility, asynchronous systems

1 Introduction

Dynamic distributed systems based on ad-hoc collections of distributed computing devices, wireless and mobile networks, unstructured peer to peer networks, opportunistic grids or clouds are supposed to allow participants to access services and information regardless of their location, topology or mobility pattern. Nonetheless, the issue of designing reliable services which can cope with the high dynamics of these systems is a challenge.

Many reliable distributed services rely on an eventual leader election to coordinate actions. The Ω *leader detector* has been proposed as a way to implement such an abstraction [1]. It ensures that, eventually, each process in the system will be provided by an unique leader, elected among the set of correct processes, in spite of crashes, uncertainties and dynamics. However, the Ω detector cannot be implemented in a purely asynchronous system [1]. Thus, some additional assumptions on the underlying system should be made in order to implement it. With this aim, two orthogonal approaches can be distinguished: *timer-based* and *message-pattern*. The timer-based is the traditional approach and supposes that channels are eventually timely; the system may be described as being partially synchronous. An alternative approach assumes that the system satisfies a message exchange pattern on the execution of a communication mechanism. While the timer-based approach imposes a constraint on the physical time (to satisfy message transfer delays) the message-pattern approach imposes a constraint on the logical time (to satisfy a message delivery order) [2].

A number of leadership protocols were proposed to implement Ω. The first timer-based solutions adopted strong assumptions concerning time and channel reliability [1, 3]; afterwards, they seek to find more and more weaker conditions regarding synchrony and reliability [4–8]. Nonetheless, the totality of these protocols adopts a classical model of "known" networks in which the set of participants (Π), its cardinality (n), and maximum number of faults (f) are known.

It happens that the inherent dynamics of the new environments prevent processes from gathering a global knowledge of the system properties. The network topology is constantly changing and the best that a node can have is a local perception of these changes. Global assumptions, such as the knowledge about the whole membership, the maximum number of crashes, full connectivity or reliable communication, are no longer realistic. In these environments, message losses, failures, and partitions are common facts.

That is why recent solutions, aiming to implement Ω in a dynamic system of "unknown" networks have emerged [9–12]. They seek for models and solutions with the possible weakest assumptions, regarding the knowledge graph, the communication graph, as well as the channel connectivity and reliability, trying to get as close as possible to reality. Although these proposals lead to a breakthrough in the implementation of the leader abstraction with dynamics requirements, none of them tolerate node mobility.

Very few papers deal with node mobility [13–16]. However, for the best of our knowledge, none of them consider a system with an arbitrary graph topology that changes over time. In this paper we provide a first Ω algorithm to tolerate a generic pattern of node mobility in an unknown network, subject to messages losses and a topology that changes over time. [16] is perhaps the work with most similarity with ours. However, differently from our solution which follows a message-pattern approach, it considers a timer-based one and the existence of stable periods that should last long enough to elect a leader.

The current paper brings thus two main contributions: *(i)* The proposition of a *model to solve the leader election problem in mobile dynamic systems.* This model, although simple, captures the requirements to solve the problem and represents the network by a communication graph with a dynamic topology, in which the relations between nodes take place over a time span and moreover nodes are mobile. *(ii)* A *leadership algorithm that implements the Ω class under the proposed model.* It follows the message-pattern approach and does not assume timely links.

2 Related Work

Leadership protocols for "known" networks. A number of leadership protocols were proposed to implement Ω in an asynchronous system prone to crash failures and taking into account the classical model of "known" networks in which Π, n and f are known and moreover the communication graph is complete.

The first solutions [1, 3] adopted strong assumptions concerning reliability and time. They consider that *all* links were reliable (no message loss) and eventually timely; that is, there is an *unknown* communication bound δ and an unknown time t_0 such that, for any time $t \geq t_0$, a message sent at time t is received by time $t + \delta$. Further solutions seek to find more and more weaker conditions regarding synchrony and channel reliability. Aguilera et al. relax the strong necessity regarding the time constraints of all links, firstly proposing an algorithm in which only one process should maintain an *eventually time link* to all the other processes [4].

Afterwards, they weak the condition to an outgoing link, in such a way that one node (namely, the $\Diamond$-*source* process) should have an eventually outgoing timely link to all the other processes, while the other links may still lose messages [5, 6]. These conditions ensure that after some time only the common leader sends message forever.

Another important work in this line is due to Malkhi et al. [7] that proposes a solution without having any eventual timely links, but which considers *eventually accessible links*. Their algorithm assumes that eventually one process (namely, the $\Diamond$-*accessible* process) can send messages such that every message obtains f timely responses. One very practical interest of this assumption is that the links are moving, that is, the f responders need not to be the same and may change from one message to another. Most recently, [8] presents a solution with a weaker model that unifies the assumptions made in [5, 6] and [7]. It shows that Ω can be implemented with at least one process with f outgoing moving eventually timely links, assuming either unicast or broadcast steps.

An orthogonal and totally different approach for implementing Ω is based on the satisfaction of a *message exchange pattern* in the system. It has been proposed by [17] to implement a $\Diamond S$ failure detector and exploited so far by [14, 2] to implement Ω. They show that Ω can be built as soon as the following process behavior property (namely *eventually winning link*) is satisfied: There is a correct process p and a set Q of $(f + 1)$ processes, such that eventually the response of p from any query issued by one process $q \in Q$ is always a winning response (i.e., it is received by q among the first $(n - f)$ responses).

Leadership protocols for "unknown" networks. As told, some recent works aiming to implement Ω in a dynamic system of "unknown" networks have emerged. They seek for models with the possible weakest assumptions, regarding the knowledge and communication graph. In common, they share a reachability communication assumption between every pair of correct processes.

Jimenez et al. [9] show that it is possible to implement Ω with no knowledge about the membership of the system, even under the minimal conditions regarding link synchrony and reliability. They provide an algorithm for Ω considering an unknown network, a complete communication graph and links that are fair-lossy, but timely.

Fernandez et al. [10, 11] propose two Ω algorithms with weakest assumptions. A first algorithm considers a partial unknown network, with a global knowledge about the lower bound on the number of correct processes (represented by $\alpha = n - f$) and fair-lossy timely links. The communication graph is not complete but there are direct links between a correct process p and a set of correct processes. A second algorithm considers unknown networks and a complete communication graph. Links are fair-lossy and timely composed of output direct links between a correct process p and every correct process in the system. One important impossibility result stated by these works is the following: in an asynchronous system, where processes have no knowledge neither about α (a lower bound on the number of corrects) nor about t (a lower bound on the number of faults), any eventual leader protocol must have at least $n - f - 1$ eventually timely links.

Tucci et al. [12] studies the Ω abstraction in a system with bounded concurrency. It assumes an unknown network, but a fully connected dynamic graph. It provides the first proposal for Ω algorithms for the infinite arrival message-passing mode [18], in which an infinite number of processes may arrive and depart over time, but the number of processes which are simultaneously up is finite (including the corrects).

Leadership protocols with node mobility. [13], Masum et al. present an Ω algorithm which, contrarily to ours, assumes totally reliable and timely channels.

Cao et al. [14] provide an implementation of Ω for a network composed of mobile hosts (MH) and mobile support stations (MSS). The eventual leader is an MH, but it is elected by the MSSs. Differently from our work, the set of MSS forms a static distributed system of reliable channels in a "known" network.

Melit et al. [15] propose both a model and an Ω algorithm that tolerate node mobility and partitions. But, to converge, their approach requires that the topology eventually does not change. Unlike to ours, this last requirement prevents arbitrary changes in the topology along the system existence.

In [19], the authors propose an Ω specification suited to dynamic systems where processes can leave and join the system, as well as an eventually timely based algorithm that implements such a specification. Gomez-Calzado et al. [16] extended the specification that takes into account graph joins/fragmentations and process mobility, proposing also a new algorithm. They make a stability assumption to converge, in which there are no graph partitioning and the existence of bidirectional connectivity among processes. Differently from our solution, they adopt a timely assumption during stable periods and some other conditions in the graph.

3 Model for Eventual Leader Election in Mobile Systems

The system is a collection of mobile nodes which communicate by sending and receiving messages via a network with broadcast facilities. There are no assumptions on the relative speed of processes or on message transfer delays, thus the system is *asynchronous*. To simplify the presentation, we take the range $\mathcal{T}$ of the clock's tick to be the set of natural numbers. There is no global clock and processes do not have access to $\mathcal{T}$: it is introduced for the convenience of the presentation and make proofs.

3.1 Communication Model

Time-Varying Communication Graph. The network is represented by a communication graph with a dynamic topology, thus the relations between nodes take place over a time span $\mathcal{T} \subseteq \mathbb{N}$. Following [20], we consider that the dynamics of the network is represented by a *time-varying graph*, namely TVG.

Definition 1. *[Time-varying graph]. A* TVG *is a tuple* $\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta, \psi)$*, where: (1)* $V = \Pi$ *represents the set of nodes, (2)* $E \subseteq V \times V$ *represents the set of communication links between nodes, (3)* $\mathcal{T} \subseteq \mathbb{N}$ *is a time span, (4)* $\rho : E \times \mathcal{T} \rightarrow \{0, 1\}$ *is an edge presence function, indicating whether a given edge* $e \in E$ *is available at a given time* $t \in \mathcal{T}$*, such that* $\rho(e, t) = 1$ *iff* e *is present at* t*, otherwise* $\rho(e, t) = 0$*, (5)* $\zeta : E \times \mathcal{T} \rightarrow \mathbb{N}$ *is a latency function, indicating the time taken to* *cross a given edge e if starting at a given time t* [3]*; (6)* $\psi : V \times \mathcal{T} \to \{0,1\}$ *is a node presence function, indicating whether a given process* $p_i \in V$ *is up at a given time* $t \in \mathcal{T}$*, such that* $\psi(p_i, t) = 1$ *iff node* p_i *is up at* t*, otherwise* $\psi(p, t) = 0$.

We use the notation $e_{i,j} = (p_i, p_j)$ for the edge between p_i and p_j. We denote N_i^t to be the set of 1-hop *neighbors* of p_i and E_i^t to be the set of edges that connect p_i to these neighbors at time $t \in \mathcal{T}$. The neighborhood relationship establishes the edge set, in such a way that $p_j \in N_i^t$ *iff* $e_{i,j} \in E_i^t$, such that $\rho(e_{i,j}, t) = 1$. The *degree* of p_i at time t is defined to be $Deg_i^t = |E_i^t|$. Given a TVG $\mathcal{G}$, the graph $G = (V, E)$ is called the *underlying graph* of $\mathcal{G}$. G should be considered as a sort of *footprint* of $\mathcal{G}$ which flattens the time dimension and indicates only the pair of nodes that have relations at some time in $\mathcal{T}$. *Journeys* can be thought of as paths over time from a source to a destination.

Definition 2. *[Journey] A sequence of couples* $\mathcal{J} = \{(e^1, t_1), (e^2, t_2), \ldots, (e^k, t_k)\}$*, such that* $\{e^1, e^2, \ldots, e^k\}$ *is a walk in* G*, is a journey in* $\mathcal{G}$ *if and only if* $\rho(e^i, t_i) = 1$ *and* $t_{i+1} \geq t_i + \zeta(e^i, t_i)$ *for all* $i < k$*. Let departure(*$\mathcal{J}$*)* $= t_1$ *be the starting date and arrival(*$\mathcal{J}$*)* $= t_k + \zeta(e^k, t_k)$ *be the last date of the journey. Let* $\mathcal{J}_{(i,j)}$ *be a journey from* p_i *to* p_j*; in this case, we say that* p_i *reaches* p_j *or more simply,* $p_i \rightsquigarrow p_j$*. Let us denote by* $\mathcal{J}_{\mathcal{G}}^*$ *the set of all possible journeys in* $\mathcal{G}$*, and by* $\mathcal{J}_{(i,j)}^* \subseteq \mathcal{J}_{\mathcal{G}}^*$ *those journeys starting at* p_i *and ending at* p_j*.*

Channels. Local broadcast between 1-hop neighbors is *fair-lossy*. This means that messages may be lost, but, if a correct p_i broadcasts m to processes in its neighborhood an infinite number of times, then every p_j permanently in the neighborhood receives m from p_i an infinite number of times, otherwise p_j is faulty or out of p_i's neighborhood. That is, if p_i starts to send m at time t an infinite number of times, then, if $\rho(e_{i,j}, t') = 1, \forall t' \in [t; +\infty)$, p_j receives m an infinity number of times if p_j is a correct neighbor of p_i. In the case of a wireless network, this condition is e.g. attained if the MAC layer reliably delivers broadcast data, even in presence of unpredictable behaviors, such as fading, collisions, and interference; solutions in this sense were proposed in [21, 22].

3.2 Process Model

We consider the *finite arrival model* [18]: the network is a dynamic system composed of infinitely many mobile processes; but each run consists of a finite set Π of n nodes, namely, $\Pi = \{p_1, \ldots, p_n\}$.

The membership is unknown. Processes are not aware about Π or n, because, moreover, these values can vary from run to run [18]. There is one process per node; each process knows its own identity, but it does not necessarily know the identities of the others. A process may fail by *crashing*, *i.e.*, by prematurely or by deliberately halting (switched off); a crashed process does not recover. Indeed, a process can re-connect to the system, but with a new identity, thus, it is considered as a new process. Processes may re-connect as they wish, but the number of re-entries is bounded, due to the finite arrival assumption. Until it possible crashes,

[3] Note that the effective delivery of a message sent at time t on an edge could be subjected to further constraints regarding the latency function, such as the condition that $\rho(e)$ returns 1 for the whole interval $[t; t + \zeta(e, t))]$.

a process behaves according to its specification. A process that does follow its algorithm specification and never crashes is said to be *correct*.

Let us thus define the status that a process may exhibit along the system execution. Informally, a *stable* process is a correct process that never leaves the system; otherwise, it is *faulty*.

Definition 3. *[Process status]. Let $t \in \mathcal{T}$. A process p_i may assume the following status.*
$stable^t(p_i) \Leftrightarrow \forall t' \geq t,\ \psi(p_i, t') = 1$
$faulty^t(p_i) \Leftrightarrow (\exists s, s < t,\ \psi(p_i, s) = 1) \wedge (\forall t' \geq t, \psi(p_i, t') = 0)$

The *failure pattern* of the system, namely $F(t)$, is the set of processes that have failed in the system by time t. That is, $F(t) = \{p_i : faulty^t(p_i)\}$. Similarly, $S(t)$, is the set of processes that are stable in the system by time t. That is, $S(t) = \{p_i : stable^t(p_i)\}$.

Definition 4. *[Process sets]. The set of processes in the system may be divided into:* $\textsc{Stable} \stackrel{def}{=} \bigcup_{t \in \mathcal{T}} S(t)$ *and* $\textsc{Faulty} \stackrel{def}{=} \bigcup_{t \in \mathcal{T}} F(t)$

3.3 The Ω Class

A *leader* oracle is a distributed entity that provides processes with a function *leader*() that when invoked by p outputs a single process q, denoted the leader. In the context of a dynamic system, a leader oracle of the Ω class satisfies the following Eventual leadership property: *There is a time after which every stable process always trusts the same stable process.* Therefore, the *leader*() function ensures that eventually the same leader is trusted by all stable processes in the system; moreover the leader is stable. Nonetheless, no process knows when such an election took place.

3.4 Network Connectivity

To solve the eventual leader abstraction, we are mostly interested in the *transmission* TVG induced by the stable nodes in the system.

Definition 5. *[Transmission TVG]. The transmission* TVG *is a tuple $\mathcal{G}_S^{tr} = (V_S, E_S, \mathcal{T}, \rho_{tr}, \zeta, \psi)$, in which $V_S = \textsc{Stable}$; $E_S \subseteq V_S \times V_S$ and ρ_{tr} is a transmission edge presence function: $\rho_{tr}(e_{i,j}, t) = 1$ iff a message sent from p_i at time t is delivered to and handled by p_j at time $t + \zeta(e_{i,j}, t)$.*

We can identify classes of TVG based on the temporal properties established by the entities. The classes are important because they imply necessary conditions and impossibility results for distributed computations. Notably, Class 5 (Recurrent connectivity) [20] is important to our study. It means that, at any point t in time, the TVG $\mathcal{G}_S^{tr}$ remains connected over time. Thus, for all stable nodes p_i, p_j, at any time, $p_i \leadsto p_j$.

Assumption 1 *[Network recurrent connectivity]. In the subsystem of stable nodes, represented by TVG $\mathcal{G}_S^{tr}$, $\forall p_i, p_j \in V_S$, $\forall t \in \mathcal{T}$, $\exists \mathcal{J} \in \mathcal{J}^*_{(p_i,p_j)}$: departure($\mathcal{J}$) $> t$.*

The recurrent connectivity is a fundamental assumption, mandatory to ensure reliable dissemination of messages to all stable processes in a dynamic network [20] and thus to ensure the properties of the leader oracle [1, 9, 23].

4 An Eventual Leader Oracle for Mobile Systems

4.1 Stable Query-Response Communication Mechanism

Our eventual leader oracle solution is based on the *message pattern approach* [17] and uses, to this end, a local QUERY-RESPONSE communication mechanism [23] adapted to a network with unknown membership. At each *query-response* round, a node systematically broadcasts a QUERY message to the nodes in its neighborhood until it possibly crashes or leaves the system. The interval between two consecutive queries is finite but arbitrary. Each couple of QUERY-RESPONSE messages is uniquely identified in the system. A process p_i launches the primitive by sending a QUERY(m) with a message m. When a process p_j delivers this query, it updates its local state and systematically answers by sending back a RESPONSE(m') with a message m' to p_i. Then, when p_i has received at least α_i responses from different processes, the current QUERY-RESPONSE *terminates*. Without loss of generality, the response for p_i itself is among the α_i responses.

Formally, the QUERY–RESPONSE primitive has the following properties:

(i) QR-Validity: If a QUERY(m) is delivered by process p_j, it has been sent by process p_i;

(ii) QR-Uniformity: A QUERY(m) is delivered at most once by a process;

(iii) QR-Termination: Let t be the time at which a process p_i terminates to send a query. If $faulty^t(p_i)$ does not hold, then that query generates at least α_i RESPONSE(m') messages from a subset of X_i processes, $|X_i| \geq \alpha_i$.

An implementation of a couple of QUERY-RESPONSE communication over fair-lossy local channels can be done by the repeated broadcast of the query by the sender p_i until it has received at least α_i responses from its neighbors. Since the communication pattern followed is local, α_i is defined locally as a function of the expected number of stable known neighbors with whom p_i may communicate at the time t in which the QUERY is issued. We consider that f_i is the maximum number of faulty processes in p_i's neighborhood. Thus, since the set of responses received by p_i includes its own response, $\alpha_i = |N_i^t| - f_i + 1$, which guarantees the liveness of QUERY-RESPONSE rounds. To ensure that at least one stable node p_j ($p_j \neq p_i$) receives the QUERY and sends a response to p_i, $\alpha_i > f_i + 1$.

The local choice for α_i changes from existing solutions which consider a global value either proportional to the total number of correct processes [17] or the total number of stable processes [23] or the total number of faults [14] in the system. Moreover, it follows recent works on fault tolerant communication in radio networks which propose a "local" fault model, instead of a "global" fault model, as an adequate strategy to deal with the dynamics and unreliability of wireless channels in spite of failures [22]. To reliably delivery data in spite of crashes, the maximum number of local failures should be $f_i < Deg_i^t/2$ [24, 25].

The following property holds:

Property 1. **Stable Termination Property ($\mathcal{S}at\mathcal{P}$).** Let p_i be a node which issues a QUERY. Thus, $\exists p_j \in$ STABLE$, p_j \neq p_i$, which receives that QUERY.

For the leadership problem, the *stable termination* is necessary for the reliable dissemination of the information to the whole network and consequent satisfaction of the properties. It is a guarantee that information from/to p_i is going to be sent/received to/from at least a stable p_j in its neighborhood. Moreover, it ensures that the first QUERY issued by p_i, when it joins the network, will be delivered by at least one stable process in such a way that p_i may take part to the membership of the system.

4.2 Behavioral Property

Instead of synchrony assumptions, to ensure the accuracy of the election, we have adopted a *message pattern* model which establishes conditions on the logical time the messages are delivered by processes. These are unified in the *stabilized responsiveness property* or $\mathcal{SRP}$.

Property 2. **Stabilized Responsiveness Property ($\mathcal{SRP}$).** Let X_j^t be the set of processes from which p_j has received responses to its last QUERY sent before t. Process p_i satisfies $\mathcal{SRP}$ at time t:

$$\mathcal{SRP}^t(p_i) \text{ iff } stable^t(p_i) \wedge \forall p_j \in \Pi \ (\exists e_{i,j}, \exists t' \geq t, \rho_{tr}(e_{i,j}, t') = 1) \Rightarrow \forall t'' \geq t' + \zeta(e_{i,j}, t'), p_i \in X_j^{t''}$$

$\mathcal{SRP}^t(p_i)$ states that there exists a time t after which all nodes of p_i's neighborhood receive, to every of their queries, a response from p_i which is always among the first α_j responses to the query. Similarly to the winning channel approach, defined in [2], the response of p_i is always a winning response. In other words, $\mathcal{SRP}^t(p_i)$ denotes the ability of a stable node p_i to eventually always reply, among the first α_j nodes, to a QUERY sent by p_j. In this case, the channel between p_i and p_j is an eventually winning channel. Moreover, as nodes may move, the $\mathcal{SRP}^t(p_i)$ states as well that neighbors of p_i eventually stop moving outside p_i's neighborhood.

To solve Ω, the $\mathcal{SRP}(p_i)$ property should hold for one stable node p_i in the system; thus preventing a probable leader p_i to be permanently demoted. As a matter of comparison, in the timer-based model, this property would be: there is a time t after which the output channels from a stable node p_i to every other neighbor p_j that communicates with p_i are eventually timely.

4.3 An Eventual Leader Election Algorithm

Algorithm 1 describes a protocol for implementing Ω in a mobile system satisfying the model, properties, and assumptions stated in Sections 3 and 4.

Notations. The algorithm uses the following variables and functions:

- mid_i: a counter used to timestamp every couple of QUERY-RESPONSE messages. Before broadcasting a new QUERY, p_i increments mid_i. These two operations are atomically performed.
- $local\_known_i$: the current knowledge of p_i about its neighborhood, i.e., the set of nodes that communicated directly with p_i. It is composed of tuples of the form $\langle mid_j, p_j \rangle$: mid_j is associated with the greatest timestamp value of a QUERY or RESPONSE message received by p_i from p_j.
- $global\_known_i$: the current knowledge of p_i about the membership of the system. Similarly to $local\_known_i$, it is composed of tuples of the form $\langle mid_j, p_j \rangle$.
- $punish_i$: a set of tuples of the form $\langle ct, p \rangle$ where ct is a punish counter and p the identity of the punished node.

- $recvfrom_i$: the set of processes that replied to the last QUERY of p_i.
- $MaxKnown()$: a boolean function that checks if p_i has the greatest timestamp associated to a message received from p_j. It is used to verify if a given neighbor process has moved or not.
- $UnionMax(set_1, set_2, ...)$: a function that performs the union of sets whose tuple elements have the form $\langle ct, p\rangle$. If $\langle -, p\rangle$ belongs to several sets, the function considers the one whose value ct is the greatest one.
- $Update\_State()$: a function used to update the state of p_i's sets with the most recent information. It keeps the tuples $\langle ct, p\rangle$ with the greatest counters in these sets. It is used to evaluate the contents of a receiving message (QUERY or RESPONSE).
- $leader()$: function that returns the current leader.

Underlying principle. The algorithm elects the leader on a basis of a punishment procedure and on the periodic exchange of QUERY-RESPONSE messages. Processes exchange these messages to know each other, to show that they are alive, as well as to share the necessary information to elect the leader. If a QUERY sent by process p_i is not responded by a process p_j that p_i locally knows, then p_j is punished by p_i. Each time p_i punishes p_j it increments the counter ct_j associated to p_j in $punish_i$.

The rationale behind the punishment procedure is that a process that fail will be infinitely often punished. The algorithm thus will eventually elect a stable process that has the smallest punish counter. To ensure that all the nodes will elect the same leader, processes should exchange their information regarding locally known processes and their respective punishment counters. Thus, each QUERY or RESPONSE message sent by p_i, beyond the message id (mid_i), carries the sets $punish_i$ and $global\_known_i$. Since the network remains connected over time (Assumption 1), the information exchanged will achieve all stable processes.

To tolerate the mobility of nodes, the algorithm makes use of the message counters. The timestamp of the last message received from processes is used to avoid false suspicions in case of mobility. If p_j is in $local\_known_i$ and if it moves from p_i's neighborhood, then it will be punished by p_i according to the last message received. But, as soon as p_i gets the information (by the contents of a received message) that another node has received a message from p_j with a greater timestamp, p_i stops to punish p_j. In this case, p_i suspects p_j to have moved from its neighborhood and considers that it is still alive in the network.

Description. Initially, p_i sends a first QUERY to introduce itself to the network (line 8). Then, four tasks are launched: $T1$, $T2$, $T3$ and $T4$.

Task $T1$ *[Punishment]*: This task is made up of an infinite loop. At each round, process p_i waits for at least α_i responses, which includes p_i's own response. For each RESPONSE(mid_j, $punish_j$, $global\_known_j$) not received from p_j such that p_j is considered as a current neighbor of p_i (i.e., it belongs to $local\_known_i$) and p_j is not suspected to have moved from p_i's neighborhood (i.e., p_i has a greater message timestamp received from p_j than the other processes of which p_i is aware), then p_j will be punished by p_i (lines 15 – 19). Notice that if it is the first time that p_j is punished by p_i, then, its punish counter will be equal to $\langle c_{min} + 1, p_j\rangle$ (line 17). Then, mid_i counter is incremented and a QUERY(mid_i, $punish_i$, $global\_known_i$) message is sent to all nodes in p_i's neighborhood.

Algorithm 1 Eventual Leader Election for Mobile Networks

`Init:`

  $punish_i \leftarrow \{\langle 0, i \rangle\}$
  $local\_known_i \leftarrow \{\langle mid_i, i \rangle\}$
  $global\_known_i \leftarrow \{\langle mid_i, i \rangle\}$
  $recvfrom_i \leftarrow \emptyset$
  $mid_i \leftarrow 1$
  `broadcast` QUERY$(mid_i, punish_i, global_known_i)$

`Task T1: [Punishment]`
`Repeat forever`
  `Wait until` $|recvfrom_i| \geq \alpha_i$
  { *Punishing known processes which did not responded* }
  `If` $\forall p_j : \langle -, p_j \rangle \in local\_known_i \wedge p_j \notin recvfrom_i \wedge MaxKnown(p_j)$ `then`
    `If` $\langle 0, p_j \rangle \in punish_i$ `then`
      $c_{min} \leftarrow min\, c : \langle c, - \rangle \in punish_i$
      `replace in` $punish_i$ $\langle 0, p_j \rangle$ `by` $\langle c_{min} + 1, p_j \rangle$
    `Else`
      `replace in` $punish_i$ $\langle v, p_j \rangle$ `by` $\langle v + 1, p_j \rangle$
  $recvfrom_i \leftarrow \emptyset$
  $mid_i \leftarrow mid_i + 1$
  `broadcast` QUERY$(mid_i, punish_i, global_known_i)$
`End repeat`
`Task T2: [Response]`
`upon reception of` RESPONSE $(mid_j, punish_j, global\_known_j)$ `from` p_j

  $UpdateState(mid_j, punish_j, global\_known_j, p_j)$
  $recvfrom_i \leftarrow recvfrom_i \cup \{p_j\}$

`Task T3 [Query]`
`upon reception of` QUERY $(mid_j, punish_j, global\_known_j)$ `from` p_j

  $UpdateState(mid_j, punish_j, global\_known_j, p_j)$
  `send` RESPONSE $(mid_i, punish_i, global\_known_i)$ `to` p_j

`Task T4 [Leader Election]`
`upon the invocation of` *`leader()`*

  `return` l `such that` $\langle c, l \rangle = Min(punish_i)$

`MaxKnown` (p) `[Max counter associated with p]`

  `If` $x : \langle x, p \rangle \in local\_known_i \geq y : \langle y, p \rangle \in global\_known_i$ `then`
    `return true`
  `Else`
    `return false`

`UpdateState` $(mid_j, punish_j, global\_known_j, p_j)$ `[Union of states]`

  $local\_known_i \leftarrow UnionMax(local\_known_i, \{\langle mid_j, p_j \rangle\})$
  $global\_known_i \leftarrow UnionMax(global\_known_i, global\_known_j, \{\langle mid_j, p_j \rangle\})$
  $punish_i \leftarrow UnionMax(punish_i, punish_j)$

Task $T2$ *[Response]*: In this task, node p_i handles the reception of a RESPONSE message sent by p_j containing mid_j, as well as the sets $punish_j$ and $global\_known_j$. Process p_i calls upon $Update\_State()$ to update its state about punishment of processes, membership, and neighborhood with more recent information coming from p_j. It also includes p_j with the respective mid_j in the set of processes that answered to its last query ($local\_know_i$), as well as in the set that keeps its membership knowledge ($global\_known_i$).

Task $T3$ *[Query]*: In this task, node p_i handles the reception of a QUERY message sent by p_j containing mid_j, as well as the sets $punish_j$ and $global\_known_j$. Similarly to $T2$, node p_i updates its state about punishment of processes, membership, and neighborhood with more recent information coming from p_j. It also answers p_j's QUERY with a RESPONSE message timestamped with its mid_i counter.

Task $T4$ *[Leader]*: This task handles the invocation of $leader()$. Whenever called, the $leader()$ function returns the process with the smallest counter in $punish_i$, thanks to the $Min(punish_i)$ function (line 39). In the case that more than a node satisfies such a condition, the identities of the nodes break the tie. Eventually, all nodes will elect the same leader, as proved in the next section.

5 Proof of Correctness

We present a sketch of proof that Algorithm 1 ensures the *eventual leadership* property. We consider a mobile network of unknown membership that satisfies the model and assumptions stated in Sections 3 and 4.

Notations.

(i) The *state* of a process p_i in time t is represented by the contents of its variables at t.

(ii) Let set_i be one of the sets $local\_known_i$, $global\_known_i$ or $punish_i$ of process p_i. We denote set_i^t this set at time t. Moreover, $set_i^t(p_j) = c$ if the value $\langle c, p_j \rangle \in set_i$ at time t; otherwise $set_i^t(p_j) = \bot$. We denote $set_i^*(p_j)$ as the set of all values of $set_i^t(p_j)$ such that $t \in \mathcal{T}$ and $set_*^*(p_j)$ as the set of all $set_i^*(p_j)$, $i \in \Pi$.

(iii) Let m be a message sent by p_i. Then, m is either a QUERY or a RESPONSE message and it contains mid_i and the sets $punish_i$ and $global\_known_i$.

(iv) We consider that process p_j *punishes* p_i if it executes lines 17 or 19 increasing the counter of p_i in its $punish_j$ set.

(v) Let us denote the set SBP as the subset of processes that have a bounded value on the punish set of all processes, $SBP = \{p_i \in \Pi \mid punish_*^*(pi) \textit{ is bounded}\}$.

Lemma 1. *Let $\mathcal{J}_{(i,j)}$ be a journey from p_i to p_j in the TVG $\mathcal{G}_S^{tr}$. Let t_0 be the departure and t_f be the arrival of $\mathcal{J}_{(i,j)}$. Let set be either punish or global_known. For any process p_k, if $set_i^{t_0}(p_k) \neq \bot$ then $set_j^{t_f}(p_k) \neq \bot \wedge set_j^{t_f}(p_k) \geq set_i^{t_0}(p_k)$.*

Proof. We first show that the lemma holds for the one-step journey $\mathcal{J}_{(i,j)} = \{(e_{i,j}, t_0)\}$, i.e., there is a message m sent by p_i at time t_0 which is delivered and handled by p_j at time $t_f = Arrival(\mathcal{J}_{(i,j)})$. We denote $punish_m$ and $global\_known_m$ the sets $punish_i^{t_0}$ and $global\_known_i^{t_0}$ carried by m respectively. Upon reception of m, p_j calls $UpdateState()$ and the result of $UnionMax(set_j, set_m, ...)$ is stored in set_j. Thus, after m is handled, if $set_i^{t_0}(p_k) \neq \bot$, then $set_j^{t_f}(p_k) \neq \bot \wedge$

$set_j^{t_f}(p_k) \geq set_i^{t_0}(p_k)$. Moreover, $punish_i$ is modified either (i) when p_i punishes some process or (ii) upon reception of m. In (i), $punish_i$ is updated in line 17 and 19. In both cases, the associated counter values are increased by at least one. In (ii), the result of $UnionMax(punish_i, punish_k)$ is stored in $punish_i$. Therefore, values in $punish_i$ never decrease locally. On the other hand, $global\_known_i$ is only updated on reception of a message and, thus, similarly to $punish_i$, values in $global\_known_i$ never decrease as well. Since in a journey, the arrival of a message precedes the departure of the message that follows, by induction and transitivity of inequality, the lemma holds for a journey of any step size.

Observation 1. Let $mid_i = c$ at time t. If a process p_j does not receive any message sent by p_i after t then $local\_known_j^{t'}(p_i) \leq c$ or $local\_known_j^{t'}(p_i) = \perp$, $\forall t' \geq t$. This follows since $local\_known_j(p_i)$ is updated by p_j upon the reception of a message from p_i and, from assumption, the value mid_m carried by this message is such that $mid_m \leq c$.

Lemma 2. *Let p_i be a stable process and $t \in \mathcal{T}$. If $\mathcal{SRP}^t(p_i)$ then there is a time $u \geq t$ after which no process punishes p_i.*

Proof. Let p_j be a process. Three cases are possible.

Case 1: p_j is faulty. If $faulty^u(p_j)$, $u \geq t$, then p_j will not punish p_i after u.

Case 2: p_j is stable and it receives a message sent from p_i at time $t' > t$. Since $\mathcal{SRP}^t(p_i)$ holds and $t' > t$, $\forall u \geq t' + \zeta(e_{i,j}, t')$ $p_i \in X_j^u$. Thus, after u, because $p_i \in recvfrom_j$, the predicate of line 14 will always return false and p_j will never punish p_i after u.

Case 3: p_j is stable and it never receives a message from p_i, sent after t. In this case, (i) either p_j does not receive any message from p_i or (ii) p_j receives at least one message from p_i. In (i), if p_j never receives a message from p_i at any time, the latter will never be added to the set $local\_known_j$. Therefore, the predicate of line 14 always returns false and p_j never punishes p_i. In (ii), if p_j receives at least one message from p_i, then p_i sent this message at time t at the latest. Let $mid_i = c$ at time t. Due to Observation 1, $local\_known_j^t(p_i) \leq c$. As p_i is stable, there is a time $t' > t$ such that $mid_i = c + 1$ and p_i broadcasts a QUERY. Upon reception of its own RESPONSE at time $t'' > t'$, p_i updates its local state. In particular $global\_known_i^{t''}(p_i)$ is updated to $c + 1$ (line 51). Furthermore, Assumption 1 (recurrent connectivity) ensures that there is a journey $\mathcal{J}_{(i,j)}$ from p_i to p_j, such that $departure(\mathcal{J}_{(i,j)}) > t''$ and $arrival(\mathcal{J}_{(i,j)}) = u$. According to Lemma 1, $global\_known_j^u(p_i) \geq global\_known_i^{t''}(p_i) = c + 1$. Thus, $\forall u' \in T,\ u' > u \Rightarrow global\_known_j^{u'}(p_i) > c \geq local\_known_j^t(p_i)$ and, thus, every call to $Maxknown()$ will always return false. It follows then that after u, p_j never punishes p_i.

We have shown that for any process p_j, there is a time u after which p_j never punishes p_i. As there is a finite number of processes, there is a finite time after which no process punishes p_i.

Lemma 3. *Let p_i be a process such that no process punishes p_i after a finite time t. Thus, $p_i \in SBP$.*

Proof. Since after t, no process punishes p_i, a process p_j punishes p_i at most the number of times p_j broadcasts a query till t. As there is a finite number of processes (from the finite arrival assumption), over all processes, the overall total number of times p_i is punished is finite. Let pun_i be this number and let max_pun_i be the maximum value by which the punish counter of p_i is incremented or updated $\forall p_j \in \Pi$ (note that at each punish step, the counter associated to p_i is either incremented by 1 at line 19 or set to $c_{min}+1$ at line 17). Then, as the initial value of every punish counter is 0, we have $\forall s \in T, \forall p_j \in \Pi, punish_j^s(p_i) \leq pun_i * max\_pun_i \ \vee \ punish_j^s(p_i) = \bot$; and, by definition of SBP, $p_i \in SBP$.

Lemma 4. *Let $p_i \in SBP$. There is a time t after which p_i is not punished by any process.*

Proof. The proof is by contradiction. Let us assume that $\forall t \in T$, $\exists (t', p_j) \in T \times \Pi$, such that $t' > t$ and p_j punishes p_i at time t'. Hence, process p_i is punished infinitely often and, as the number of processes is finite, there is a process p_j that punishes p_i infinitely often. It follows, therefore, that $punish_j^*(p_i)$ is not bounded, which is a contradiction.

Theorem 1. *SBP is the set of processes that are eventually not punished.*

Proof. Theorem 1 follows directly from lemma 3 and lemma 4.

Lemma 5. *Let $p_j \in$ Faulty. p_j will be punished an infinite number of times by at least one process $p_i \in$ Stable. Thus, it follows that $SBP \subset$ Stable.*

Proof. When p_j connects to the system, it broadcasts at least one query, corresponding to the first message sent upon execution of line 8. Let $faulty^t(p_j)$ and $last\_mid_j$ be the last value of mid_j before t. Since the increment of variable mid_j and the query (lines 7–8 or 21–22) are performed atomically (i.e., p_j does not crash between these two operations), p_j broadcasts a query with $mid_j = last\_mid_j$ before crashing. Furthermore, due to the stable termination Property 1 ($\mathcal{S}at\mathcal{P}$), there is at least one process $p_i \in$ Stable that receives this query. Thus, there is a time t' such that $local\_known_i^{t'}(p_j)$ and $global\_known_i^{t'}(p_j)$ equal to $last\_mid_j$.

We remark (lines 50 and 51) that no process p_k inserts in its $global\_known_k$ set neither in its $local\_known_k$ set the tuple $\langle mid_j, p_j \rangle$, such that $mid_j > last\_mid_j$, since $last\_mid_j$ is the greatest value of mid_j of any message received from p_j. Thus, after t', each call by process p_i to the function $MaxKnown(p_j)$ returns true. Let be $t'' = max(t, t')$. Since $stable^{t''}(p_i)$, the number of queries sent by p_i after t'' is infinite. Moreover, since p_j crashed at time $t \leq t''$, p_j does not respond to any of those queries. Therefore, p_i will punish p_j infinitely often.

Lemma 6. *Let $p_j \notin SBP$ be a process such that $\exists p_i$, $p_i \in$ Stable which punishes p_j infinitely often. Then, $\forall p_k \in$ Stable, $punish_k^*(p_j)$ is unbounded.*

Proof. Since p_i punishes p_j infinitely often, $punish_i^*(p_j)$ is unbounded. Let $p_k \in$ Stable, $p_k \neq p_i$. Let us show that $punish_k^*(p_j)$ is unbounded as well. Let $b \in \mathbb{N}$, since $punish_i^*(p_j)$ is unbounded, there is a time $t \in T$ such as $punish_i^t(p_j) \geq b$. From Assumption 1 (recurrent connectivity) there is a journey $\mathcal{J}_{(i,k)}$ from p_i to p_k, such that $t' = departure(\mathcal{J}_{(i,k)}) > t$ and $arrival(\mathcal{J}_{(i,k)}) = t''$. As punish values increase over time and according to Lemma 1, $punish_k^{t''}(p_j) \geq punish_i^t(p_j) > b$. We conclude then that $punish_k^*(p_j)$ is unbounded.

Lemma 7. *Let $p_i \in SBP$. There is a time t after which $\forall p_j, p_j \in$ STABLE will carry the same $punish_j^t(p_i)$ value for p_i and this value never changes after t.*

Proof. Since $p_i \in SBP$, $\exists b \in \mathbb{N}$, such that $\forall s \in T, \forall p_j \in \Pi$, $punish_j^s(p_i) < b \vee punish_j^s(p_i) = \perp$. This remains true if $p_j \in$ STABLE. Furthermore, there is a time when p_i adds itself to $punish_i$ (line 3). Thus, $punish_*^*(p_i) \neq \emptyset$ and it is bounded. As $punish_*^*(p_i)$ is composed of integer values, there exists a maximum value; let $max\_punish(p_i)$ be such a maximum value. Let p_j be the stable process such that $punish_j^s(p_i) = max\_punish(p_i)$. Due to Assumption 1 (recurrent connectivity), there is a journey $\mathcal{J}_{(j,k)}$ from p_j to p_k, such that $departure(\mathcal{J}_{(j,k)}) > s$ and $arrival(\mathcal{J}_{(j,k)}) = s'$, $s' > s$. On the one hand, following Lemma 1, $punish_k^{s'}(p_i) \geq max\_punish(p_i)$. On the other hand, since $punish_k^{s'}(p_i) \leq max\_punish(p_i)$, we conclude that $punish_k^{s'}(p_i) = max\_punish(p_i)$. Moreover, the punish values increase over time. Thus, $\forall s''$, $s'' \geq s' \Rightarrow punish_k^{s''}(p_i) = max\_punish(p_i)$. Since there is a finite number of stable processes, $\forall p_k \in$ STABLE, there is a time s'_k where $punish_k^{s'_k}(p_i) = max\_punish(p_i)$. Let be $t = max(s'_k | p_k \in$ STABLE$)$ then $\forall p_k \in$ STABLE, $\forall t' \geq t$, $punish_k^{t'}(p_i) = max\_punish(p_i)$.

Theorem 2. *Algorithm 1 satisfies the eventual leadership property.*

Proof. From assumption, there is at least one process $p_i \in$ STABLE satisfying $\mathcal{SRP}^s(p_i)$ at time s. According to Lemma 2, $p_i \in SBP$; thus, $SBP \neq \emptyset$. According to Lemma 7 and the finite arrival assumption, $\exists t \in T, \forall t' > t, \forall p_i \in SBP, \forall p_j \in$ STABLE, $punish_j^{t'}(p_i) = max\_punish(p_i)$. Let $maxSBP = Max(max\_punish(p_k))$, $p_k \in SBP$. From Lemma 6, the finite arrival assumption and the fact that the punish values never decrease, $\exists t'' \in T$, $\forall p_j \in$ STABLE, $\forall p_k \notin SBP, \forall t' > t''$ $maxSBP < punish_j^{t'}(p_k)$. Thus, there exists a time $u = max(t, t'')$ after which $Min(punish_j)$ will return the same tuple $\langle c, p_i \rangle$, $\forall p_j$, such that $p_i \in SBP$. Hence, upon invoking the $leader()$ function after u, all stable processes will return the same process identity as the leader.

6 Conclusion

This paper has provided a model and an algorithm to solve the eventual leader election problem in mobile dynamic systems, in which both the network topology and relations between mobile nodes evolve over time. The algorithm implements the Ω class, following the message-pattern approach and exploiting the TVG framework to represent the dynamics of the network topology. As a future research, we plan to extend the results by also considering the timer-based approach.

References

1. Chandra, T., Toueg, S.: Unreliable failure detectors for reliable distributed systems. JACM **43**(2) (March 1996) 225–267
2. Mostefaoui, A., Raynal, M., Travers, C.: Time-free and timer-based assumptions can be combined to obtain eventual leadership. IEEE TPDS **17**(7) (2006) 656–666
3. Larrea, M., Fernandez, A., Arévalo, S.: Optimal implementation of the weakest failure detector for solving consensus. In: SRDS 2000. (2000) 334–334
4. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: Stable leader election. In: Proc. of the 15th Int. Conf. on Distributed Computing. (2001) 108–122

5. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: On implementing omega with weak reliability and synchrony assumptions. In: PODC 2003, ACM Press (2003) 306–314
6. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: Communication-efficient leader election and consensus with limited link synchrony. In: PODC 2004. (July 2004) 328–337
7. Malkhi, D., Oprea, F., Zhou, L.: Meets paxos: Leader election and stability without eventual timely links. In: DISC 2005. (2005) 199–213
8. Hutle, M., Malkhi, D., Schmid, U., Zhou, L.: Chasing the weakest system model for implementing ω and consensus. IEEE Transactions on Dependable and Secure Computing **6** (2009) 269–281
9. Jiménez, E., Arévalo, S., Fernandez, A.: Implementing unreliable failure detectors with unknown membership. Inf. Process. Lett. **100**(2) (2006) 60–63
10. Fernandez, A., Jimnez, E., Raynal, M.: Eventual leader election with weak assumptions on initial knowledge, communication reliability, and synchrony. In: DSN. (2006) 166–175
11. Fernandez, A., Jimnez, E., Raynal, M.: Eventual leader election with weak assumptions on initial knowledge, communication reliability, and synchrony. Journal of Computer Science and Technology **25**(6) (2010) 1267–1281
12. Tucci-Piergiovanni, S., Baldoni, R.: Eventual leader election in infinite arrival message-passing system model with bounded concurrency. In: EDCC 2010. (2010) 127 –134
13. Masum, S.M., Ali, A.A., Touhid-youl Islam Bhuiyan, M.: Asynchronous leader election in mobile ad hoc networks. In: AINA Conference. (2006) 827–831
14. Cao, J., Raynal, M., Travers, C., Wu, W.: The eventual leadership in dynamic mobile networking environments. In: PRDC Conference. (2007) 123–130
15. Melit, L., Badache, N.: An Ω-based leader election algorithm for mobile ad hoc networks. In: 4th Networked Digital Technologies Conf. (2012) 483–490
16. Gomez-Calzado, C., Larrea, M., Raynal, M.: Fault-tolerant leader election in mobile dynamic distributed systems. Technical report, University of the Basque Country UPV/EHU (2013)
17. Mostefaoui, A., Mourgaya, E., Raynal, M.: Asynchronous implementation of failure detectors. In: DSN Conference. (2003) 351–360
18. Aguilera, M.K.: A pleasant stroll through the land of infinitely many creatures. SIGACT News **35**(2) (2004) 36–59
19. Larrea, M., Raynal, M., Soraluze, I., Cortiñas, R.: Specifying and implementing an eventual leader service for dynamic systems. Int. J. Web Grid Serv. **8**(3) (2012) 204–224
20. Casteigts, A., Flocchini, P., Quattrociocchi, W., Santoro, N.: Time-varying graphs and dynamic networks. In: Adhoc-Now Conference. (2011) 346–359
21. Min-Te, S., Lifei, H., A. Arora, A., L.Ten-Hwang: Reliable mac layer multicast in ieee 802.11 wireless networks. In: ICPP Conference. (2002) 527–536
22. Koo, C.Y.: Broadcast in radio networks tolerating byzantine adversarial behavior. In: PODC 2004. (2004) 275–282
23. Mostefaoui, A., Raynal, M., Travers, C., Patterson, S., Agrawal, D., Abbadi, A.: From static distributed systems to dynamic systems. In: SRDS 2005. (2005) 109–118
24. Bhandari, V., Vaidya, N.H.: On reliable broadcast in a radio network. In: PODC 2005, ACM (2005) 138–147
25. Bhandari, V., Vaidya, N.H.: Reliable broadcast in radio networks with locally bounded failures. IEEE TPDS **21** (2010) 801–811