Best-effort Group Service in Dynamic Networks

Bertrand Ducourthial, Sofiane Khalfallah, Franck Petit

Best-effort Group Service in Dynamic Networks*

Bertrand Ducourthial[‡] Sofiane Khalfallah[‡]
Franck Petit[†]

[‡] (1) Université de Technologie de Compiègne
(2) CNRS Heudiasyc UMR6599
Centre de Recherche de Royallieu
B.P. 20529, Compiègne, France

[†] (1) Université PM. Curie
(2) CNRS LIP6
Paris, France

Abstract

We propose a group membership service for dynamic ad hoc networks. It maintains as long as possible the existing groups and ensures that each group diameter is always smaller than a constant, fixed according to the application using the groups. The proposed protocol is self-stabilizing and works in dynamic distributed systems. Moreover, it ensures a kind of *continuity* in the service offer to the application while the system is converging, except if too strong topology changes happen. Such a *best effort* behavior allows applications to rely on the groups while the stabilization has not been reached, which is very useful in dynamic ad hoc networks.

Keywords: Group maintenance, Best effort, Stabilization, Dynamic network

1 Introduction

Self-stabilization in dynamic networks. A *dynamic* network can be seen as an (*a priori* infinite) sequence of networks over time. In this paper, we focus on dynamic *mobile* networks. Examples of such networks are *Mobile Ad hoc* networks (MANETs) or *Vehicular Ad hoc* networks (VANETs).

Designing applications on top of such networks require to deal with the lack of infrastructure [20, 15]. One idea consists in building virtual structures such as clusters, backbones, or spanning trees. However, when the nodes are moving, the maintenance of such structures may require more control. The dynamic of the network increases the control overhead. Thus, distributed algorithms should require less overall organization of the system in order to remain useful in dynamic networks.

Another paradigm for building distributed protocols in mobile ad hoc networks consists in designing self-stabilizing algorithms [4]. These algorithms have the ability to recover by themselves (*i.e.*, automatically) from an inconsistent state caused by transient failures that may affect a memory or a message. A topology change can be considered as a transient failure because it leads to an inconsistency in some memories. Indeed, when a node appears or disappears in the network, all its neighbors should update their neighborhood knowledge.

*Supported by Région Picardie, proj. APREDY.

Self-stabilizing algorithms have been intensively studied the two last decade for their ability to tolerate transient faults [8]. However, it is important to notice that such algorithms do not ensure all the time the desirable behavior of the distributed system, especially when faults occur and during a certain period of time following them. In dynamic systems, it becomes illusory to expect an application that continuously ensures the service for which it has been designed. In other words, what we can only expect from the distributed algorithms is to behave as "the best" as possible, the result depending on the dynamic of the network.

In this paper, we propose a new approach in the design of distributed solutions for dynamic environments. We borrow the term "*best-effort*" from the networking community to qualify the algorithms resulting of our approach. Roughly speaking, a best-effort algorithm is a self-stabilizing algorithm that also maintains an extra property, called *continuity*, conditioned by the topology changes.

Continuity aims to improve the output of the distributed protocol during the convergence phase of the algorithm, provided that a topological property is preserved. This means that there is a progression in the successive outputs of the distributed protocols, except if the network dynamic is too high. This is important in a distributed system where the dynamic (that is, the frequent topology changes) can prevent the system to converge to the desirable behavior. Since the output of the protocol will certainly be used before the stabilization, the continuity ensures that third party applications can rely on it instead of waiting. The output will certainly be modified in the future, but without challenging previous ones.

In some aspects, our approach is very close to the ones introduced in [16] and in [9]. In [16], the authors introduce the notion of *safe-convergence* which guarantees that the system quickly converges to a safe configuration, and then, it gracefully moves to an optimal configuration without breaking safety. However, the solution in [16] works on a static network. In [9], the authors use the notion of *passage predicate* to define a *superstabilizing* system, *i.e.*, a system which is stabilizing and when it is started from a legitimate state and a single topology change occurs, the passage predicate holds and continues to hold until the protocol reaches a legitimate state. By contrast, the continuity property is intended to be satisfied *before* a legitimate configuration has been reached. It must be satisfied during the stabilization phase, and between two consecutive stabilization phases (convergence phase followed by stability phase).

We illustrate our approach with a new group management protocol adapted to vehicular ad hoc networks (VANET), an emblematic case of dynamic ad hoc networks.

Dynamic group membership service in VANET. The Intelligent Transportation Systems (ITS) currently attract a lot of attention. It is expected that such systems could improve the road safety, offer a better resource usage, increase the productivity, reduce the impact of transport on the environment. ITS is extensively studied by both theoretical and experimental researchers, especially the vehicular ad hoc networks (VANET), showing characteristics that are different from many generic MANETs [3].

Many VANET applications require cooperation among close vehicles during a given period: collaborative driving, distributed perception, chats and other infotainment applications. Vehicles that collaborates form a *group*. A group is intended to grow until a limit depending on the application. For instance, the distributed perception should not involve too far vehicles, a chat should be responsive enough, that limits the number of hops, etc. When the group diameter is larger than the bound given by the application, it should be split into several smaller groups.

However, a group should not be split if this is not mandatory by the diameter constraint in order to ensure the best duration of service to the application relying on it. Even if another partitioning of the network would have been better (*e.g.*, less groups, no isolated vehicle), it is preferable to maintain the composition of existing groups. It is expected that, thanks to the mobility of the nodes, small groups will eventually succeed in merging. It is then more important to maintain existing groups as long as possible.

To achieve this specific *group membership service*, we propose a self-stabilizing distributed algorithm in a wireless communication model. This algorithm stabilizes the views in such a way that all the members of a group will eventually share the same view (in which only the members appear). The groups' diameters are smaller than a fixed applicative constant `Dmax` and neighbor groups merge while the diameter constraint is fulfilled. Moreover, our algorithm admits the following continuity property: no node disappears from a group except if a topology change leads to the violation of the diameter constraint (or a node leaves).

To the best of our knowledge, only a few number of papers addresses the problem of group membership maintenance in the context of self-stabilization. Recently, in [6], the authors propose a self-stabilizing k-clustering algorithm for static networks. In [10], the authors propose a self-stabilizing group communication protocol. It relies on a mobile agent that collects and distributes information during a random walk. This protocol does not allow to build groups limited to k hops.

Group communication structures have been proposed in the literature to achieve fault-tolerance in distributed systems [2], by providing for instance replication, virtual synchrony, reliable broadcast, or atomic broadcast (*e.g.*, [19, 14]). Other works deal with the k-clustering or k-dominating set problem, *e.g.*, [5, 1, 17, 18, 16], where nodes in a group are at most at distance k from a *cluster-head* or *dominant node*. The aim of these algorithms is to optimize the partitioning of the network. The group service we propose in this paper is different in the sense that its aim is not to optimize any partitioning nor to build group centered to some nodes. Instead, it tries to maintain existing groups as long as possible while satisfying a constraint on the diameter, without relying on a specific node (that may move or leave).

Contributions. In Section 2, we describe the distributed system we consider in this paper. We also state what it means for a protocol to be self-stabilizing and best effort regarding a continuity property conditioned by topology changes. Next, in Section 3, we specify a new group service (inspired from VANET). In the same section, our best-effort self-stabilizing algorithm is presented. The proofs are given in Section 4. Finally, we make some concluding remarks in Section 5. By lake of place, some proofs are in appendix.

2 Model

We consider a system $\mathcal{S}$ composed of mobiles nodes that communicate by wireless communication devices.

Node. Let V be a set of nodes spread out in an Euclidean space. The total number of nodes in V is finite but unknown. Each node is equipped with a processor unit (including local memory) and a wireless communication device. A node can move in the Euclidean space. A node u has either the state *active* or *passive*. If it is active, a node u can compute, send and

receive messages by executing a local algorithm. The *distributed* protocol $\mathcal{P}$ is composed of all the local algorithms.

Communication. We define the *vicinity* of a node v as the part of the Euclidean space from where a node u can send a message that can be received by v (the vicinity depends on the communication devices, the obstacles, etc.). A node v can receive a message from u if (i) both u and v are active, (ii) u is in the vicinity of v, (iii) u is sending a message, (iv) no other node in the vicinity of v is currently sending a message, and (v) v is not sending a message itself (any active node that is not sending is able to receive).

We admit the following *fair channel* hypothesis: there exists two time constants τ_1 and τ_2 with $\tau_1 \geq \tau_2$ such that, starting from a date t, any node v is able to receive before the date $t + \tau_1$ a message from each node u, providing that u is in the vicinity of v between t and $t + \tau_1$ and attempts to send a message every τ_2 units of time. As in [10], we use timers rather than an asynchronous distributed algorithm for discovering the neighborhood because our approach is intended to dynamic networks with too frequent changes for considering an asynchronous distributed algorithm.

At any time instant t, there is a *communication link* from u to v if both u and v have the state active (at t), and if u is into the vicinity of v (at t). A communication link is oriented because u could be in the vicinity of v while the converse is false. The capacity of a communication link is one message.

Dynamic. Since nodes can move and change their states, the topology of the system $\mathcal{S}$ evolves over time. Even if a communication link exists from u to v at date t, a communication may fail because previous conditions are not fulfilled, or because the duration of the link is too short.

The dynamic of a network is in fact a relative notion, depending on both the dynamic of the nodes (moving and state changing) and the speed of the messages. Indeed, even if the nodes move slowly, if the communication are very slow, some distributed applications may fail. Conversely, the nodes could move rapidly without disturbing distributed applications if communications are efficient. We then introduce the following metric of dynamic.

The system $\mathcal{S}$ is *δ-dynamic* if any node u experimenting a neighborhood change is able to send a message to all the nodes v at distance smaller than or equal to δ before a new topology change occur (note that during this message propagation, the topology is fixed). In the rest of this paper, we admit the following hypothesis: $\mathcal{S}$ is 1-dynamic.

Executions. A *configuration* c of $\mathcal{S}$ is the union of the states of memories of all the processors and the contents of all the communication links. An empty communication link is denoted in the configuration by a link that contains an empty set of messages; obviously a non existing communication link is not reported in the configuration (this is important to tackle topology changes). Let $\mathcal{C}$ be the set of configurations.

An *execution* of a distributed protocol $\mathcal{P}$ over $\mathcal{S}$ is a sequence of configurations $c_0, c_1, \ldots$ of $\mathcal{S}$ which (i) does not contain successive identical configurations ($\forall i \in \mathbb{N}$, $c_{t_i} \neq c_{t_{i+1}}$), (ii) contains all the successive configurations the system $\mathcal{S}$ reached by executing the distributed protocol $\mathcal{P}$, providing that at least one node has noticed the change and (iii) is either infinite, or the computation is finite, no action is enabled and no message is in transit in the final configuration (this implies that links remain stable).

By consequence, any topology change detected by at least one node leads to a new configuration. There is then a single topology per configuration; we denote by G_i the topology of $\mathcal{S}$ during the i-th configuration. In a *static* system $\mathcal{S}$, we have $G_i = G_0$ in every execution $c_0, c_1, c_2, \ldots$. Otherwise, the system $\mathcal{S}$ is said to be *dynamic*.

Self-Stabilization. Let $\mathcal{X}$ be a set. Then $x \vdash \Pi$ means that an element $x \in \mathcal{X}$ satisfies the predicate Π defined on the set $\mathcal{X}$ and $X \vdash \Pi$ with $X \subset \mathcal{X}$ means that any $x \in X$ satisfies $x \vdash \Pi$. We define a special predicate true as follows: $\forall x \in \mathcal{X}$, $x \vdash \mathsf{true}$. Let Π_1 and Π_2 be two predicates defined on the set of configurations $\mathcal{C}$ of the system $\mathcal{S}$. Π_2 is an *attractor* for Π_1 if and only if the following condition is true: for any configuration $c_1 \vdash \Pi_1$ and for any execution $e = c_1, c_2, \ldots$, there exists $i \geq 1$ such that for any $j \geq i$, $c_j \vdash \Pi_2$.

Define a *specification* of a task as the predicate Π on the set $\mathcal{C}$ of configurations of system $\mathcal{S}$. A protocol $\mathcal{P}$ is self-stabilizing for Π if and only if there exists a predicate $\mathcal{L}_{\mathcal{P}}$ (called the legitimacy predicate) defined on $\mathcal{C}$ such that the following conditions hold:
1. For any configuration $c_1 \vdash \mathcal{L}_{\mathcal{P}}$, and for any execution $e = c_1, c_2, \ldots$, we have $e \vdash \Pi$ (correctness).
2. Π is an attractor for true (closure and convergence).

Best effort continuity of service. We denote by Π_T a *topological* predicate defined on the pairs of successive configurations in an execution. Such a predicate is intended to be false when an "important topology" change happens. We denote by Π_C a *continuity* predicate defined on the pairs of successive configurations in an execution. Such a predicate is intended to be false when the quality of the outputs produced by protocol $\mathcal{P}$ in the two successive configurations decreases.

The protocol $\mathcal{P}$ offers a best effort continuity of services if $\Pi_T \Rightarrow \Pi_C$.

3 Dynamic Group Service

In this section, we first state the group membership service considered in this paper. Next, we present an algorithm for this problem, followed by its proof.

3.1 Specification

Let $G(V, E)$ be a graph. Let $d(u, v)$ be the *distance* between u and v (length of the shortest path from u to v in G). A *subgraph* $H(V_H, E_H)$ is defined as follows: $V_H \subseteq V$ and $\forall (u, v) \in E$, $(u \in V_H \text{ and } v \in V_H) \Rightarrow (u, v) \in E_H$. Two subgraphs $H_1(V_1, E_1)$ and $H_2(V_2, E_2)$ of a graph G are said *distinct* if $V_1 \cup V_2 = \emptyset$. Let $X \subseteq V$ be a set of nodes. We denote by $d_X(u, v)$ the distance between u and v in the subgraph $H(X, E_H)$, that is, the length of the shortest path from u to v with only edges of E_H. If such a path does not exists, then $d_X(u, v) = +\infty$.

Given a graph G, the problem considered in this paper consists in designing a distributed protocol that provides a partition of G into disjoint subgraphs called *groups* that satisfies constraints described below.

Denote by $\mathtt{view}_v^c$ the knowledge of v about its group in configuration c (output on node v). Let Π_A be the predicate defined on the configurations and called *agreement property*: $\Pi_A(c)$ holds if and only if there exists a partition of disjoint subgraphs $H_1(V_1, E_1), H_2(V_2, E_2), \ldots,$

$H_i(V_i, E_i), \ldots$ of $G(V, E)$ such that for every pair u, v, $u \in V_i$ and $v \in V_i \Leftrightarrow \texttt{view}_u^c = \texttt{view}_v^c = V_i$.

Let Ω_v^c be the *group* of v in configuration c, defined by: (i) $\Omega_v^c = \texttt{view}_v^c$ if $v \in \texttt{view}_v^c$ and $\forall u \in \texttt{view}_v^c, \texttt{view}_v^c = \texttt{view}_u^c$, (ii) $\Omega_v^c = \{v\}$ otherwise. Note that given any configuration c, if $\Pi_A(c)$ holds, $\{\Omega_v^c, v \in V\}$ defines a partition of G into disjoint subgraphs of G, *i.e.*, there exists a partition of disjoint subgraphs $H_1(V_1, E_1), H_2(V_2, E_2), \ldots, H_i(V_i, E_i), \ldots$ such that $\forall v \in V_i,\ \Omega_v^c = V_i$ for every subgraph H_i.

Let `Dmax` be an integer representing the maximal admissible distance between two nodes belonging to the same group. Let Π_S be the predicate defined on the configurations and called *safety property*: $\Pi_S(c)$ holds if each group is connected and its diameter is smaller than `Dmax`. More formally $\Pi_S(c) \equiv \forall v \in V$, $\max_{x,y \in \Omega_v^c} d_{\Omega_v^c}(x, y) \leq \texttt{Dmax}$.

Let Π_M be the predicate defined on the configurations and called *maximality property*: $\Pi_M(c)$ holds if by merging two existing groups, we cannot obtain a partition satisfying the safety property. More formally $\Pi_M(c) \equiv \forall u, v \in V$ with $\Omega_u^c \neq \Omega_v^c$, $\exists x, y \in \Omega_u^c \cup \Omega_v^c$, $d_{\Omega_u^c \cup \Omega_v^c}(x, y) > \texttt{Dmax}$.

The problem considered in this paper is to design a self-stabilizing protocol regarding predicates $\Pi_S \wedge \Pi_S \wedge \Pi_M$: after the last failure or topology change, the algorithm converges in finite time to a behavior where Π_A, Π_S, and Π_M are fulfilled.

Note that the above requirement is suitable for fixed topologies only. The following predicate deals with dynamic system, *i.e.*, with topological change of the network. Let $G^c(V^c, E^c)$ be the graph modeling the topology of the system at configuration c. We introduce the following notation: d^c refers to the distance in the graph G^c, and $d_X^c(u, v)$ denotes the distance between u and v in G^c by considering only edges of the subgraph $H(X, E_H)$ of G^c. Define the *topological property* as the predicate Π_T defined on any couple of two successive configurations c_i, c_{i+1} of an execution e as follows: $\Pi_T(c_i, c_{i+1})$ holds if, for any pair of nodes belonging to the same group in c_i, the distance between them will still be smaller than `Dmax` in c_{i+1}. In other words, if a topology change occurred between c_i and c_{i+1}, it has preserved the maximal distance condition. More formally, $\Pi_T(c_i, c_{i+1}) \equiv \forall v \in V$, $\max_{x,y \in \Omega_v^{c_i}} d_{\Omega_v^{c_i}}^{c_{i+1}}(x, y) \leq \texttt{Dmax}$.

Finally, we are looking for protocols attempting to preserve a group partition when a topology change occurs. Let Π_C be the predicate defined on the couples of successive configurations and called *continuity property*: $\Pi_C(c_i, c_{i+1})$ holds if in any group, no node disappears. In other words, an application can work with the given view because it defines a group in which no node will disappear. More formally, $\Pi_C(c_i, c_{i+1}) \equiv \forall v \in V$, $\Omega_v^{c_i} \subset \Omega_v^{c_{i+1}}$. Obviously, if the dynamic of the network is too large, such a property cannot be satisfied. We then introduce the best effort requirement: $\Pi_T \Rightarrow \Pi_C$.

3.2 Distributed Protocol

3.2.1 Informal description

For a given node, the candidates to form a group are neighbors up to distance `Dmax`. Nodes build lists of candidates by diffusing messages in the neighborhood (see under). Only symmetric links are taken into account. In $O(\texttt{Dmax})$ the knowledge of the `Dmax` neighborhood can be known. Malformed lists are rejected (such as lists larger than `Dmax`). Moreover, when a node receives a list which is too long compared to its current list, it rejects it to avoid any split of its current group.

When a node enters in a new group, its arrival will be propagated to the group's members

in $O(\texttt{Dmax})$. Such an arrival can increase the diameter of the group. A new member will be accepted only if the diameter constraint is respected. Two nodes could be accepted concurrently by two distant members of a given group and the diameter constraint is no more fulfilled for this group. In this case, one of the new member must leave the group (instead of splitting the existing group). To avoid any inopportune change in the views, a new member enters in the view of a node after the end of its *quarantine* period. This allows to guarantee that its arrival has been approved by all the members (no conflicts).

When a node has to leave the group to fulfill the diameter constraint, the choice is done using *priority* (function `pr`). Priorities are totally ordered; if $\texttt{pr}(u) < \texttt{pr}(v)$, then u has the priority. We do not detail `pr` in this paper. This may be the identity of the nodes, but dynamic priorities are more useful. For instance, if priorities are based on nodes' identities, a new member with a small identity may split an old group just after entering it because it moves too far (consider a group of vehicles in a stable convoy and a rapid vehicle that overtakes the convoy). To the contrary, if priorities are based on the logical date of entrance into the group (implemented using time-stamps), then this is the last arrived node that will leave (the vehicles that overtakes).

Priorities on the nodes allow to easily define priorities on the groups by taking the smallest priority of the members[1]. Priorities on the groups allows to ensure the merging of neighbor groups (and the maximality property Π_M) in particular cases (loop of groups willing to merge).

3.2.2 Building the lists of ancestor sets

The messages sent to the neighbors contain *ordered list of ancestors' sets*. The *ordered list of ancestors' sets* of a node v is defined by: $(a_v^0,\ a_v^1,\ \ldots,\ a_v^p)$ where any node $x \in a_v^i$ satisfies $d(x,v) = i$ $(a_v^0 = \{v\})$ and p is the distance of the farthest ancestor of v.

Computations are done using the r-operator ant [7, 13, 11]. Let $\mathbb{S}$ be the set of lists of vertices' sets. For instance, if a, b, c, d, e are vertices, $(\{d\},\{b\},\{a,c\})$ and $(\{c\},\{a,e\},\{b\})$ belong to $\mathbb{S}$. Let $\oplus$ be the operator defined on $\mathbb{S}$ that merges two lists while deleting needless or repetitive information (a node appears only one time in a list of ancestors' sets). For instance, $(\{d\},\{b\},\{a,c\}) \oplus (\{c\},\{a,e\},\{b\}) = (\{d,c\},\{b,a,e\},\{a,c,b\}) = (\{d,c\},\{b,a,e\})$. Finally, let r be the endomorphism of $\mathbb{S}$ that inserts an empty set at the beginning of a list. For instance, $r(\{d\},\{b\},\{a,c\}) = (\emptyset,\{d\},\{b\},\{a,c\})$. We then define the operator ant by: $\text{ant}(l_1,l_2) = l_1 \oplus r(l_2)$, where l_1 and l_2 are lists belonging to $\mathbb{S}$. This is a strictly idempotent r-operator [11] inducing a partial order relation. It leads to self-stabilizing static tasks (building the complete ordered lists of ancestor sets) in the register model [13]. Since our wireless communication model admits bounded links, these results can be extended to this model. (Refer to the discussion related to r-operators in wireless networks in [7].)

3.2.3 Algorithm

The distributed protocol *Dynamic Group Service* is composed of a single algorithm per node (uniform protocol), see below. Each node computes its output (`list`, `view` and priorities) when its timer T_c expires. It broadcasts its output in the neighborhood when the timer T_s expires $(T_s \leq T_c)$. Timers T_c and T_s are chosen according to the *fair channel hypothesis*. In case both T_c and T_s expire simultaneously, we assume that the action related to T_c are performed

[1]During the stabilization, the topology is supposed to be fixed and then no node leaves. The dynamicity is taken into account thanks to the best effort property.

before those of T_s (computation before sending) in order to avoid any fairness problem (no computation).

All messages received from the neighborhood are collected in `msgSet`. If a neighbor sends more than one message before the timer expiration, only the last received is kept. After computation, the variable `msgSet` is reset in order to detect when a neighbor leaves.

Algorithm *Dynamic Group Service* (node `v`)

```
1   Upon reception of a message msg sent by a node u:
2     update message of u in msgSet
3   Upon T_c timer expiration:
4     compute()
5     reset msgSet
6     restart timer T_c with duration τ_1
7   Upon T_s timer expiration:
8     send( list_v with priorities, priority of view_v ) to the neighbors
9     restart timer T_s with duration τ_2
```

A computation (in procedure `compute`, below) consists in building the ordered list of ancestor' sets as well as the view. The list is sent to the neighbors to be used in their ant computation. The view is the output of the protocol used by the third party applications (eg. chat, collaborative perception...) which requested the dynamic group service, and which gave the diameter constraint `Dmax` (fixed during all the execution).

First, the incoming lists are checked. Line 3, when the list sent by u and received by v does not contain v, is malformed or is too long[2], it is replaced by $(\underline{u})$. When u receives the list of v containing $\underline{u}$, it accepts the list of v and sends a list containing v. Thanks to this triple handshake, the link has been detected as symmetric (by the way, asymmetric link information are not propagated).

Line 6, if the received list is too long, the sender u is marked as incompatible ($\underline{\underline{u}}$). Roughly speaking, a list received by a node u from another node v is compatible if, by combining its list with the one of v, u does not increase the diameter of its group beyond `Dmax`. In order to reach this goal, it is enough to test if the sum of the lengths of both lists is less than or equal to $\texttt{Dmax}+1$. But, such simple test would avoid to merge two groups by taking advantage of short cuts between both groups. In other words, this would ignore the knowledge that nodes of a group have on nodes belonging to the other group. The technical condition used in Function `compatibleList()` deals with such an optimization.

Line 9, if the sender is external to the group, the priorities of the nodes inside its lists are replaced by the priority of its view (received with its list Line 8): inter-groups comparisons are realized using groups' priorities instead of nodes' priorities.

Then a first computation is performed using the ant operator (Lines 13-16; this computation ought to be performed inside the first `forall` loop but we preferred to separate it for clarity). Thanks to the `goodList` test, the size of the incoming lists are smaller than $\texttt{Dmax}+1$. However, the computed list could reach the size of $\texttt{Dmax}+2$ while the maximum is $\texttt{Dmax}+1$ (the ant operation increases by one the list sizes). In this case, a choice has to be done between either the local node v or the farthest nodes in the received lists. This choice is done by using the priority (function `pr`, Line 19). If the local node v has not the priority on the too far node w (positions in the list start from 0 to $\texttt{Dmax}+1$ here), the list in which w appears are ignored

[2] $s(\texttt{list})$ returns the number of elements in `list`; $\texttt{list}.i$ returns the ith element of `list`, starting from 0.

(Line 22). At the opposite side of the group, node w keeps the list containing v but the end of its ordered list of ancestor's sets will be truncated (meaning that v and w will not belong to the same group). Indeed, after the too far nodes have been all examined, the list of ancestors is computed again (Lines 27-30) and is truncated (Line 31) in order to delete the too far nodes (these remaining too fare nodes have less priority than v).

In order to not include a node in a view while it could be rejected later, a *quarantine mechanism* is used. The quarantine period of a node willing to enter in a group is fixed at `Dmax` timers. Each time a computation is done (and then the new node progresses in the group), its quarantine period decreases. Since the group diameter is less than or equal to `Dmax`, any conflict would have been detected before the new node enters into a view. Moreover, if a member of the group accepts the new node, then all the members will accept it. The procedure `compute()` is given below.

Procedure compute()

▷ *Checking the received lists*

1 **for all** $\texttt{list}_u$ in `msgSet` **do**
2 delete marked nodes except $\underline{v}$ in $\texttt{list}_u$ ▷ *Marked nodes are only useful between neighbors.*
3 **if** $\neg$ goodList($\texttt{list}_u$) **then** ▷ *List of u cannot be used;*
4 replace $\texttt{list}_u$ by $(\underline{u})$ in `msgSet` ▷ *this list is ignored but the sender is kept.*
5 **end if** ▷ *Now, incoming lists cannot be larger than* `Dmax`.
6 **if** $u \notin \texttt{view}_v$ **and** $\neg$ compatibleList($\texttt{list}_u$) **then** ▷ *u is new, but its list cannot be accepted;*
7 replace $\texttt{list}_u$ by $(\underline{\underline{u}})$ in `msgSet` ▷ *u is denoted as an incompatible neighbor*
8 **end if**
9 **if** $u \notin \texttt{view}_v$ **then** ▷ *If the sender is external, using group priorities.*
10 update priorities in $\texttt{list}_u$ with priority of $\texttt{view}_u$
11 **end if**
12 **end for**

▷ *Computing the list of ancestors' sets of v.*

13 $\texttt{list}_v \leftarrow (v)$
14 **for all** $\texttt{list}_u \in$ `msgSet` **do**
15 $\texttt{list}_v \leftarrow$ ant($\texttt{list}_v, \texttt{list}_u$) ▷ *Computation using the* ant *r-operator.*
16 **end for**

▷ *Removal of incoming lists containing too far nodes (after* ant *computation,* $\texttt{list}_v$ *cannot be larger than* $\texttt{Dmax}+1$*)*

17 **if** $s(\texttt{list}_v) = \texttt{Dmax}+2$ **then** ▷ *The list is too long.*
18 **for all** w at position $\texttt{Dmax}+1$ in $\texttt{list}_v$ **do** ▷ *Scanning too far nodes.*
19 **if** $\texttt{pr}(w) < \texttt{pr}(v)$ **then** ▷ *Far node w has the priority.*
20 **for all** $\texttt{list}_u \in$ `msgSet` **do** ▷ *Looking for lists that provided w;*
21 **if** w is at position `Dmax` **then** ▷ *they contain w in their last place.*
22 replace $\texttt{list}_u$ by $(\underline{\underline{u}})$ in `msgSet` ▷ *The neighbor that provided w is ignored.*
23 **end if**
24 **end for**
25 **end if**
26 **end for**

▷ *Computing* $\texttt{list}_v$ *again, without the incoming lists that contained too far nodes with priority.*

27 $\texttt{list}_v \leftarrow (v)$
28 **for all** $\texttt{list}_u$ in msgSet **do**
29 $\texttt{list}_v \leftarrow \text{ant}(\texttt{list}_v, \texttt{list}_u)$
30 **end for**
31 keeping up to $\texttt{Dmax}+1$ first elements in $\texttt{list}_v$ ▷ *Deleted too far nodes have not the priority.*
32 **end if**
33 Update quarantines: quarantine of new nodes is Dmax, non null quarantine of others decreases by 1
34 $\texttt{view}_v \leftarrow$ non marked nodes in $\texttt{list}_v$ with null quarantine
35 Update priorities ▷ *Depends on the kind of priorities used.*

Function goodList(list)
1 **if** v or $\underline{v}$ are in list.1 **and** $s(\texttt{list}) \leq \texttt{Dmax}+1$ **and** $\emptyset \notin \texttt{list}$ **then**
2 **return** true
3 **end if**
4 **return** false

Function compatibleList(list)
1 **if** $s(\texttt{list}_v) + s(\texttt{list}) \leq \texttt{Dmax}+1$ **or**
$\exists i \in \{0,\ldots,s(\texttt{list}_v)\}$, $\texttt{list}_v.i \subseteq \texttt{list}.1 \wedge \min\left(s(\texttt{list}_v)+s(\texttt{list})+1-i, s(\texttt{list})+1+i/2\right) \leq \texttt{Dmax}$
2 **return** true ▷ *Refer to Proposition 13.*
3 **end if**
4 **return** false

4 Proofs

We first focus on the self-stabilizing property of our algorithm. We show that assuming a fixed topology, the system converges in finite time to an execution satisfying the statements in Subsection 3.1, *i.e.*, $\Pi_S \wedge \Pi_A \wedge \Pi_M$ is an attractor. Next, we prove that, assuming topological changes preserving the maximal distance condition over the groups, then continuity is preserved, *i.e.*, $\Pi_T \Rightarrow \Pi_C$.

4.1 Stabilization

In the sequel, we prove that our protocol is self-stabilizing by showing that Π_S and Π_A and Π_M are attractors—Propositions 8, 7 and 12, respectively.

We begin by showing that eventually lists will become correct (Propositions 1 and 2). We first prove that any execution cannot remain infinitely with configurations having lists larger than Dmax. We denote by $e_{\texttt{Dmax}}$ the suffix of an execution e such that, for any configuration $c \in e_{\texttt{Dmax}}$, for any node $v \in V$, the size of $\texttt{list}_v$ is smaller than or equal to $\texttt{Dmax}+1$.

Proposition 1 (Dmax) *On a fixed topology, any execution e reaches in finite time a suffix e_{Dmax}.*

Starting from this proposition, we now prove that any execution cannot remain infinitely with configurations having a non existing node in a list. We denote by $e_{\texttt{exist}}$ the suffix of an execution e such that, for any configuration $c \in e_{\texttt{exist}}$, for any node $v \in V$, every node $u \in \texttt{list}_v^c$ satisfies $u \in V$.

Proposition 2 (Exist) *On a fixed topology, any execution e reaches in finite time a suffix e_{exist}.*

Next, we establish the connection between marked nodes in the algorithms and subgraphs (Propositions 3, 4, 5 and 6). We call *double-marked edge* an edge (u, v) such that either u double-marks v or v double-marks u (denoted by $\underline{\underline{u}}$ in the algorithm). The following proposition is a consequence of the double-marked edge technique. A node v double-marks its neighbor u only if the list sent by u cannot be accepted by v (Lines 7 and 22). In this case, node v will ignore the list sent by u. Reciprocally, if u has been double-marked by v, u will detect an asymmetric link (u does not appear in the list it received after Line 2) and only the identity of v will be kept by u, the rest of the list of v will be ignored (Line 4).

Proposition 3 (No propagation) *Let u and v be two vertices of G and suppose that, in any execution e, there exists a configuration c_e from which any path from u to v in G contains a double-marked edge. Then u will eventually disappear from $\texttt{list}_v^c$ and v will eventually disappear from $\texttt{list}_u^c$.*

The following proposition is a consequence of the *ant* computation (see Section 3.2.2). It propagates nodes identities (providing there is no edge-marking technique for limiting it) [13, 7].

Proposition 4 (Propagation) *Let u and v be two vertices of G and suppose that, in any execution e, there exists a configuration c_e from which there exists a path from u to v in G without double-marked edge. Then $\texttt{list}_v^c$ will eventually contain u and $\texttt{list}_u^c$ will eventually contain v.*

Proposition 5 (Double-marked edge) *Suppose that $d(u, v) > \texttt{Dmax}$. Then any execution admits a suffix $e_{\texttt{edge}}$ such that, for any configuration $c \in e_{\texttt{edge}}$, there is a double-marked edge on any path from u to v.*

Let denote by $H_v^c(V_{H_v}, E_{H_v})$ the subgraph of $G(V, E)$ defined in the configuration c by: for any node u in V_{H_v}, $v \in \texttt{list}_u^c$. Such a subgraph is composed of vertices containing v in their list. We prove that eventually H_u and H_v are distinct when $d(u, v) > \texttt{Dmax}$.

Proposition 6 (Subgraphs) *Suppose that $d(u, v) > \texttt{Dmax}$. Then any execution admits a suffix $e_{\texttt{subgraph}}$ such that, for any configuration $c \in e_{\texttt{subgraph}}$, H_u and H_v are distinct subgraphs.*

The preceding propositions give the Agreement. Consider any execution $e_{\texttt{subgraphs}}$. Denote by $e_{\texttt{agree}}$ the suffix of an execution e such that $\Pi_A(c)$ holds for any configuration $c \in e_{\texttt{agree}}$, that is $V_{H_v} = \texttt{view}_w^c$ for any $w \in H_v$. The following proposition is given by Propositions 6, 4 and 3.

Proposition 7 (Agreement) *On a fixed topology, any execution e reaches in finite time a suffix $e_{\texttt{agree}}$.*

Proof. By Proposition 6, for any execution, there exists a suffix such that, for any nodes u and v in G, if $d(u, v) > \texttt{Dmax}$, then the subgraphs H_u and H_v are distinct. Consider now two nodes w and v such that w belongs to H_v

By Proposition 4, for any execution, there exists a suffix such that, for any configuration c in this suffix, the identities of H_v will be in $\texttt{list}_w^c$.

By Proposition 3, for any execution, there exists a suffix such that, for any configuration c in this suffix, the $\texttt{list}_w^c$ contains only vertices of H_v.

After the end of the quarantine period, all the nodes in $\texttt{list}_w$ belong to $\texttt{view}_w$. Then the system reaches a suffix in which all the nodes of H_v and only these nodes appear in $\texttt{view}_w$, for any vertex $w \in H_v$. Hence, $\texttt{view}_v^c = \texttt{view}_w^c = \Omega_v^c$. This gives Π_A. □

Now we have the agreement, there is a connection between subgraphs and groups. We then prove the Safety. Consider any execution $e_{\texttt{agree}}$ (Proposition 6). Denote by $e_{\texttt{safe}}$ the suffix of an execution e such that $\Pi_S(c)$ holds for any configuration $c \in e_{\texttt{safe}}$. The following proposition is a consequence of Proposition 6.

Proposition 8 (Safety) *On a fixed topology, any execution e reaches in finite time a suffix e_{safe}.*

Proof. By Proposition 6, for any execution and any nodes u and v in G satisfying $d(u,v) > \texttt{Dmax}$, the subgraphs H_u and H_v will eventually be distinct. Hence, for any execution, there exists a suffix $e_{\texttt{safe}}$ such that, for any configuration $c \in e_{\texttt{safe}}$, for any vertex v in G, $\text{Diam}(H_v^c) \leq \texttt{Dmax}$.

Then, by Proposition 7, we have $\max_{x,y \in \Omega_v^c} d_{\Omega_v^c}(x,y) \leq \texttt{Dmax}$. This gives Π_S. □

We consider any execution $e_{\texttt{agree}}$. In order to prove the maximality property, we introduce the following definitions. An edge (u,v) is *internal* in a given configuration c if $\Omega_u^c = \Omega_v^c$. In the converse case ($\Omega_u^c \neq \Omega_v^c$), it is *external*. An external edge involves double-marked nodes and it is then not propagated by the algorithm (marked nodes are deleted, see line 2 in Procedure `compute()`). We denote by *nee* (resp. *ndg*) the function defined on $\mathcal{C}$ that returns the number of external edges in a given configuration (resp. the number of distinct groups in configuration c: $ndg(c) = |\{\Omega_v^c, v \in V\}|$.

Proposition 9 *If nee is decreasing along a suffix e_s of an execution e, ndg is also decreasing along e_s.*

Proof. Let (u,v) be an external edge in a configuration c_i and assume that it is an internal edge in configuration c_{i+1}. This means that $\Omega_u^{c_i} \neq \Omega_v^{c_i}$ and $\Omega_u^{c_{i+1}} = \Omega_v^{c_{i+1}}$. Hence $nee(c_i) > nee(c_{i+1}) \Rightarrow ndg(c_i) > ndg(c_{i+1})$. □

We prove that any execution reaches in finite time a suffix in which the function *nee* does not increase. We denote by $e_{\texttt{notincr}}$ such a suffix: $\forall c_i, c_{i+1} \in e_{\texttt{notincr}}$, $nee(c_{i+1}) \leq nee(c_i)$.

Proposition 10 (Not incr.) *On a fixed topology, any execution e reaches in finite time a suffix $e_{\texttt{notincr}}$.*

Proof. Let $c \in e_{\texttt{agree}}$ be a configuration (Proposition 7). Let (u,v) be an internal edge in configuration c. Then we have $\Omega_u^c = \Omega_v^c$ and u is in $\texttt{list}_v^c$. In order (u,v) becomes an external edge, one of its extremity (say v) would have double-marked the other (in Procedure `compute()`). But this cannot happen after the `goodList` test (line 3) because $c \in e_{\texttt{subgraphs}}$. This cannot happen after the `compatibleList` test (line 6) because u is in already in $\texttt{view}_v^c$. □

Now, we prove that any execution reaches in finite time a suffix in which the function *nee* is decreasing while Π_M is not true. We denote by $e_{\texttt{decr}}$ such a suffix: $\forall c_i \in e_{\texttt{decr}}$, $\Pi_M(c_i) \vee \exists c_j \in e_{\texttt{decr}}$, $i < j$ and $nee(c_i) > nee(c_j)$.

Proposition 11 (Decreasing) *On a fixed topology, any execution e reaches in finite time a suffix* $e_{\texttt{decr}}$.

Proof. Let $c \in e_{\texttt{notincr}}$ be a configuration (Proposition 10). Starting from such a configuration, the *nee* function cannot increase. Suppose that Π_M is not true in c. Then, by definition of Π_M, there exists two neighbors nodes x and y with different views that could merge their groups without breaking Π_S. By fair channel hypothesis, a timer later the system reaches a configuration c' in which x (resp. y) has received the list sent by y (resp. x).

Without loss of generality, suppose that Ω_x has the smallest priority among all the subgraphs that can merge, and Ω_y has the smallest priority among all the groups that can merge with Ω_x.

During the `compute()` Procedure on x and y, the `goodList` tests are true because $c' \in e_{\texttt{notincr}}$ and then $c' \in e_{\texttt{safe}}$. The `compatibleList` test is true on both x and y because they cannot have change their list since configuration c. Hence we obtain: $x \in \texttt{list}_y$ and $y \in \texttt{list}_x$.

Since Ω_y has the smallest priority among the neighbors of Ω_x, no member of Ω_x can receive a message from a group with a smallest priority. Therefore x will never receive and then will never send to y a list with a too far node with a smallest priority than y one's. Hence y will never double-mark x and x will remain in the list of y.

Similarly, since Ω_x has the smallest priority among the groups that can merge, no member of Ω_y can receive a message from a group with a smallest priority. Therefore y will never receive and then will never send to x a list with a too far node with a smallest priority than x one's. Hence x will never double-mark y and y will remain in the list of x.

After `Dmax` timer, the list of y (resp. x) has reached any $u \in \Omega_x$ (resp. Ω_y) thanks to the fair channel Hypothesis. Moreover the quarantine of these new members reaches 0 and they are now included in $\texttt{view}_u$. Thus, the edge (x, y) becomes an internal edge.

Hence, starting from configuration c with $\neg\Pi_M(c)$, the system reaches in finite time a configuration c'' with $nee(c) > nee(c'')$. □

The following proposition is given by Propositions 9, 10 and 11; it shows that any execution reaches in finite time a suffix in which Π_M is true. We denote by $e_{\texttt{max}}$ such a suffix.

Proposition 12 (Maximality) *On a fixed topology, any execution e reaches in finite time a suffix* $e_{\texttt{max}}$.

Proof. By Proposition 10, the execution reaches a suffix $e_{\texttt{notincr}}$ such that the *nee* function will no more increase. By Proposition 11, the execution reaches a suffix $e_{\texttt{decr}}$ such that the *nee* function decreases while Π_M is not true. Hence, while Π_M is false, the number of external edges will eventually decrease. By Proposition 9, this means that the number of subgraphs will eventually decrease while Π_M is false. Since the graph is finite, the number of subgraphs cannot decrease infinitely and Π_M will eventually become true. □

4.2 Continuity

In this subsection, we consider the dynamic of the network. We show that if the continuity property is violated into a group, then their exists a pair of nodes belonging to that group such that the distance between them is larger that `Dmax`. The following technical proposition justifies the compatibleList test.

Proposition 13 (Compatible lists) *Let v be a node having the list $(a_v^0, a_v^1, \ldots, a_v^p)$ and assume that its neighbor w sends the list $(a_w^0, a_w^1, \ldots, a_w^q)$. Then, the diameter of the group of v will remain smaller than or equal to* `Dmax` *after v accepts w if and only if there exists $i \in \{0, \ldots, p\}$ such that w is neighbor of all the nodes belonging to a_v^i and either $p - i + 1 + q \leq$* `Dmax` *or $i/2 + q + 1 \leq$* `Dmax`*.*

Proposition 14 *For any execution e, for any configuration c_i in e, $\Pi_T(c_i, c_{i+1}) \Rightarrow \Pi_C(c_i, c_{i+1})$.*

Proof. Suppose that there exists a configuration c_i and a node v such that $\texttt{view}_v^{c_i} \not\subseteq \texttt{view}_v^{c_{i+1}}$. Then there exists a node u such that $u \in \texttt{view}_v^{c_i}$ and $u \notin \texttt{view}_v^{c_{i+1}}$. This cannot happen after u or v has added a new node in its view, thanks to the quarantine mechanism. This can only happen because either u or v removed a node from their views.

Without loss of generality, suppose that v removed a node x: $x \in \texttt{view}_v^{c_i}$ and $x \notin \texttt{view}_v^{c_{i+1}}$. If $x \notin \texttt{view}_v^{c_{i+1}}$, then (i) the quarantine of x is not null or (ii) x is not in $\texttt{list}_v^{c_{i+1}}$ or (iii) x is marked in $\texttt{list}_v^{c_{i+1}}$ (Line 34 in Procedure `compute()`).

(i) The first case is exclude because x was already in $\texttt{view}_v^{c_i}$.

(ii) In the second case, if v has not received the message of x while it received it before, then x left the neighborhood of v (fair channel Hypothesis). Moreover, x was not able to reach the neighborhood of a node w in Ω_v^c before a timer expiration on v (that guarantees the propagation up to one hop of any message) thanks to the 1-dynamic Hypothesis. Hence, in configuration c_{i+1}, there is not path from x to v with only nodes of $\Omega_v^{c_i}$ and $d_{\Omega_v^{c_i}}^{c_{i+1}}(x, v) = +\infty$. Thus $\neg\Pi_T(c_i, c_{i+1})$ (*a neighbor left*).

(iii) In the third case, if x is simple marked, its list is not good while it was in configuration c_i, which is exclude (Line 3). If x is double-marked, this cannot happen after the compatibleList test (Line 7) because x was in $\texttt{view}_v^{c_i}$. If this happened after Line 22, then x sent a list with a too far node y such that $\texttt{pr}(y) < \texttt{pr}(v)$. If $y \notin \Omega_v^{c_i}$, then $y \notin \texttt{view}_v^{c_i}$. Then the quarantine of y is not null and no node of $\Omega_v^{c_i}$ has admitted y in its view. Therefore, thanks to Proposition 13, y would have never been propagated inside $\Omega_v^{c_i}$ until v, because of the compatibleList test (Line 6). Finally, if $y \in \Omega_v^{c_i}$, then the distance from y to v in configuration c_{i+1} is larger than `Dmax`: $d_{\Omega_v^{c_i}}^{c_{i+1}}(x, v) >$ `Dmax` and $\neg\Pi_T(c_i, c_{i+1})$ (*the group stretched out*). □

5 Conclusion

This paper introduces the best effort concept to complete the self-stabilization in dynamic ad hoc networks: a continuity of service is ensured if the dynamic of the network allows it. A new group membership service inspired from VANET has been specified; its aim is to keep existing groups as long as possible and with a diameter smaller than a constant.

To achieve this specific group membership service, a self-stabilizing distributed algorithm in a wireless communication model has been designed and proved. The Dynamic Group Service stabilizes the views in such a way that all the members of a group will eventually share the same view (in which only the members appear). The groups' diameters are smaller than a fixed applicative constant `Dmax`. Neighbor groups merge while the diameter constraint is fulfilled. Moreover, this algorithm admits the following continuity property: no node disappears from a group except if a topology change leads to the violation of the diameter constraint (or a node leaves).

The protocol has been implemented and its performances are currently studied by simulation on Network Simulator, using several mobility models. We believe that the best effort approach and the continuity property are promising for building useful services on dynamic ad hoc networks.

References

[1] A.D. Amis, R. Prakash, and D.H.T. Vuaong. Max-min d-cluster formation in wireless ad hoc networks. In *IEEE INFOCOM*, pages 32–41, 2000.

[2] Kenneth P. Birman. The process group approach to reliable distributed computing. *Commun. ACM*, 36(12):37–53, 1993.

[3] J. Blum, A. Eskandarian, and L. Hoffman. Challenges of intervehicle ad hoc networks. *IEEE Transaction on Intelligent Transportation Systems,*, 5:347–351, 2004.

[4] O. Brukman, S. Dolev, Y. Haviv, and R. Yagel. Self-stabilization as a foundation for autonomic computing. In *The Second International Conference on Availability, Reliability and Security (ARES)*, pages 991–998, Vienna, April 2007.

[5] G.V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Computing Surveys*, 4(33):1–43, 2001.

[6] A. K. Datta, L. L. Larmore, and P. Vemula. A self-stabilizing O(k)-time k-clustering algorithm. *Computer Journal*, 2009.

[7] S. Delaët, B. Ducourthial, and S. Tixeuil. Self-stabilization with r-operators revisited. In *Journal of Aerospace Computing, Information, and Communication*, 2006.

[8] S. Dolev. *Self-Stabilization*. The MIT Press, 2000.

[9] S. Dolev and T. Herman. Superstabilizing protocols for dynamic distributed systems. In *Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing (PODC)*, page 255, New York, NY, USA, 1995. ACM.

[10] S. Dolev, E. Schiller, and J.L Welch. Random walk for self-stabilizing group communication in ad hoc networks. *IEEE Transactions on Mobile Computing*, 5(7):893–905, 2006.

[11] B. Ducourthial. r-semi-groups: A generic approach for designing stabilizing silent tasks. In 9^{th} *Stabilization, Safety, and Security of Distributed Systems (SSS'2007)*, pages 281–295, Paris, novembre 2007.

[12] B. Ducourthial and S. Tixeuil. Self-stabilization with r-operators. *Distributed Computing*, 14(3):147–162, 2001.

[13] B. Ducourthial and S. Tixeuil. Self-stabilization with path algebra. *Theor. Comput. Sci.*, 293(1):219–236, 2003.

[14] R. Guerraoui and A. Schiper. Software-based replication for fault-tolerance. *IEEE Transaction on Computers*, 30(4):68–74, 1997.

[15] Arshad Jhumka and Sandeep S. Kulkarni. On the design of mobility-tolerant tdma-based media access control (mac) protocol for mobile sensor networks. In Tomasz Janowski and Hrushikesha Mohanty, editors, *ICDCIT*, volume 4882 of *Lecture Notes in Computer Science*, pages 42–53. Springer, 2007.

[16] Hirotsugu Kakugawa and Toshimitsu Masuzawa. A self-stabilizing minimal dominating set algorithm with safe convergence. In *20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, 2006.

[17] S. Kutten and D. Peleg. Fast distributed construction of small-dominating sets and applications. *Journal of Algorithms*, 28(1):40–66, 1998.

[18] L. D. Penso and V. C Barbosa. A distributed algorithm to find k -dominating sets. *Discrete Applied Mathematics*, 141(1-3):243–253, 2004.

[19] F.B. Schneider. Impliementing fault tolerant services using the state machine approach: a tutorial. *Computing Surveys*, 22(4):299–319, 2990.

[20] I. Stojmenovic. *Handbook of Wireless Networks and Mobile Computings*. John Wiley & Sons, 2002.

A r-operators: a summary

When modeling the distributed algorithms with algebraic operators, interesting properties (termination, self-stabilization) can be ensured by simply checking some local properties of the operator. To stabilize a distributed algorithm while some loops exist in the network, the idempotency is required ($x \cdot x = x$). However, the operators of the idempotent semi-groups (such as $\min(x, y)$ in $\mathbb{N}$) cannot converge in presence of transient faults [12]. By using an endomorphism (such as $x \mapsto x + 1$ in $\mathbb{N}$), these operators can be generalized in *r-operators* (such as $\min(x, y + 1)$ in $\mathbb{N}$). The Abelian idempotent semi-group is then a particular case of r-semi-groups, where the endomorphism is the identity mapping $x \mapsto x$ [11]. An r-operator is r-associative ($x \triangleleft (y \triangleleft z) = (x \triangleleft y) \triangleleft r(z)$), r-commutative ($r(x) \triangleleft y = r(y) \triangleleft x$), r-idempotent ($r(x) \triangleleft x = r(x)$) and admits a left neutral element ($x \triangleleft e_{\triangleleft} = x$). Under certain conditions, an r-semi-group induces a semi-group and this gives a method to build r-operators [11] : finding an Abelian idempotent semi-group $(\mathbb{S}, \oplus)$ and then an endomorphism $r : \mathbb{S} \to \mathbb{S}$. These algebraic structures admit an order relation. An idempotent r-operator satisfies $\forall x \in \mathbb{S}, x \preceq_{\oplus} x$ where $\preceq_{\oplus}$ is the order relation of the induced semi-group. When we have $\forall x \in \mathbb{S}, x \prec_{\oplus} x$, the r-operator is *strictly idempotent*. In [7], it has been proved that the strictly idempotent r-operators that induce a total order relation lead to self-stabilizing static tasks in unreliable messages passing.

B Proofs

B.1 Proof of Proposition 1 (Dmax)

Proof. Starting from configuration c_1, the system will reach in finite time a configuration in which every node has computed its list after expiration of its timer. After such a computation, the size of the lists is bounded by $\texttt{Dmax}+1$ (because it is truncated at the $\texttt{Dmax}+1$ position, Line 31). □

B.2 Proof of Proposition 2 (Exist)

Proof. Let $c \in e_{\texttt{Dmax}}$ be a configuration (Proposition 1). Let u be a node label such that $u \notin V$ and denote by U_k^c the set of nodes having u in their list at position k in configuration c. Consider the function $\phi(c)$ defined by $\phi(c) = \min\{k \in \mathbb{N}, U_k^c \neq \emptyset\}$ and $\phi(c) = \infty$ if $\forall k \in \mathbb{N}, U_k^c = \emptyset$. We prove that ϕ is continuously growing along the execution to be eventually equal to infinity forever.

Consider a node v in $U_{\phi(c)}^c$: v contains u at position $\phi(c)$ in its computed list and no node in configuration c contains u at a smaller position in its computed list. Until the next expiration of its timer, v cannot receive a list containing u in a smaller position than $\phi(c)$. Hence, the system will reach in finite time a configuration in which the node v has computed a new list that does not contain u at a position smaller than $\phi(c)+1$. After a timer (fair channel Hypothesis), the system reaches in finite time a configuration in which the neighbors of v have received this list.

After finite time, any node $v \in U_{\phi(c)}^c$ will do the same. The system then reaches in finite time after configuration c a configuration c' in which $U_{\phi(c)}^{c'}$ is empty, meaning that $\phi(c) < \phi(c')$.

By iteration, ϕ is growing along the execution. Since the size of the lists is bounded by $\texttt{Dmax}+1$ (Proposition 1), there exists a configuration c'' reached in finite time after c in which $\phi(c'') = \infty$, meaning that u does not appear anymore in the computed lists of the nodes forever. □

B.3 Proof of Proposition 5 (Double-marked edge)

Proof. Let v and w two nodes of G such that $d(v,w) = \texttt{Dmax}+1$. Without loss of generality, we suppose that $pr(w) < pr(v)$. Suppose that there exists a path from v to w that does not contain any double-marked edge. By Proposition 4, there exists a neighbor u of v such that u sends to v a list containing w. The size of this list is larger than $\texttt{Dmax}$. There is two cases. (i) $u \notin \texttt{view}_v$. In this case, $\texttt{list}_u$ is replaced by $(\underline{\underline{u}})$. (ii) $u \in \texttt{view}_v$. In this case, v computes a list using the one sent by u. Since $d(u,v) > \texttt{Dmax}$, the resulting list is too long. Since $pr(w) < pr(v)$, the computation will be done again without the list provided by u, which will be replaced by $(\underline{\underline{u}})$. In the two cases, u is double-marked by v. Hence, any path from u to v will eventually contains a double-marked edge. □

B.4 Proof of Proposition 6 (Subgraphs)

Proof. By Proposition 5, there exists a suffix s_1 such that any path from u to v contains a double-marked edge. By Proposition 3, there exists a suffix s_2 included in s_1 such that for any configuration c in this suffix, $u \notin \texttt{list}_v^c$ and $v \notin \texttt{list}_u^c$. Then $u \notin H_v$ and $v \notin H_u$.

Let consider a node w such that $w \in H_v$ and $w \in H_u$. Then there exists at least one path from u to v containing w. The length of such a path is larger than `Dmax`. Then, by Proposition 5, it admits a double-marked edge, either on the subpath from u to w or from the subpath from w to v.

Now, let consider all the paths from u to v containing w; they all contain a double-marked edge. Suppose that for one path P_1, this double-marked edge is between w and v and for a second path P_2, it is between u and w. Then, by considering edges of P_1 from u to w and edges of P_2 from w to v, we obtain a path from u to v without any double-marked edge, which is a contradiction. Then, all paths from u to v containing w admit a double-marked edge, and this edge is always between u and w or always between w and v. Thus, w cannot belong to both H_u and H_v, meaning that there is no node w such that $w \in H_u$ and $w \in H_v$.

Hence, any execution reaches a suffix such that, for any configuration c in this suffix, H_u^c and H_v^c are distinct. □

B.5 Proof of Proposition 13 (Compatible lists)

Proof. Let $c \in e_{\texttt{safe}}$ be a configuration (Proposition 8). Let w be the first node of Ω_w^c for which the list of ancestor's sets is received by v. Then, the only external edges between Ω_v^c and Ω_w^c known by v are those joining w (external edges are not propagated). Hence, without loss of generality, assume that only these external edges exist between the groups.

($\Rightarrow$) Assume that the conditions are fulfilled. Let $u \in a_v^k$ and $u' \in a_w^l$ be two nodes in the lists of v and w respectively. There exists at most two families of shortest paths from u to u', depending on the external edge used to reach w. Let P_1 be a path that includes the edge (v, w). It starts from u and joins v by k edges in the group of v, joins w by the edge (u, v) and then reaches u' by l edges in the group of u. Let P_2 be a path from the second family. It starts from u and joins a node $v' \in a_v^i$ by $|k - i|$ internal edges in the group of v, then joins w by the edge (v', w) and then reaches u' by l internal edges in the group of u.

The length of P_1 is bounded by $k + 1 + q$. But since P_1 is a shortest path, it is shorter to reach u' from u by joining a node of a_v^0 (*i.e.*, v) than by joining a node of a_v^i (such as v'). Hence we have $k \leq i/2$ and the length of P_1 is bounded $i/2 + 1 + q$, which is smaller than `Dmax` by hypothesis. The length of P_2 is bounded by $p - i + 1 + q$, which is also smaller than `Dmax` by assumption.

Hence, for any node u and u' belonging to the group of v and w respectively, there exists a path from u to u' with less than `Dmax` edges. The list of w is then compatible with the list of v, and can then be accepted by v.

($\Leftarrow$) Assume by contradiction that the conditions are not fulfilled and that v accepts the list of w, *i.e.*, v includes the list of w by computing its new list with `ant`—refer to Lines $14-16$ of Procedure `compute()`. That means that the list of w is compatible—refer to Lines $6-8$—, which contradicts the assumption. Then the nodes of $\texttt{list}_w^c$ will be propagated in the lists of nodes of $\texttt{list}_v^c$ and reciprocally. But at least one node $u \in \texttt{list}_v^c$ will see that a node $u' \in \texttt{list}_w^c$ is too far from it and reciprocally. Either u or u' will reject the lists of its neighbors that contain the too far node (depending on the priority between u and u') and either the group of v or the group of w splits (when a neighbor is rejected by u, it disappears from $\texttt{list}_u$, and then from $\texttt{view}_u$; it is then no more in H_v). $\square$