Best-effort Group Service in Dynamic Networks

Bertrand Ducourthial, Sofiane Khalfallah, Franck Petit

Best-effort Group Service in Dynamic Networks*

(Extended Abstract)

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

Abstract

We propose a group membership service for asynchronous wireless dynamic ad hoc networks. It maintains as long as possible the existing groups and ensures that the groups' diameters are always smaller than a constant, fixed according to applications requirements. The proposed protocol is self-stabilizing in a dynamic distributed system. Moreover, it ensures a *continuity property*, meaning that, while the system is converging, it does not run away, except if it is forced by a topology change.

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 are given by *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 [8, 28]. 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 dynamicity 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 [6]. These algorithms have the ability to recover themselves from inconsistent states 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. Self-stabilizing algorithms have been intensively studied the two last decade for their ability to build robust distributed protocols [14].

It is important to notice that self-stabilization does not ensure all the time the desirable behavior of the distributed system. Indeed, after a transient fault occurs, the distributed system falls into an inconsistent state [29]. After a finite time, the protocol will ensure a legitimate behavior, according to its specification. However, each time a transient fault appears, there generally exists a *convergence period* during which the specifications are not ensured.

*Supported by Région Picardie, proj. APREDY.

†Lab. Heudiasyc UMR CNRS 6599, Univ. de Technologie de Compiègne, France

‡INRIA/LIP Lab. Univ. of Lyon/ENS Lyon, France

The duration of the convergence phase is one of the major performance parameter of self-stabilizing algorithms, called *convergence time*. Except if the convergence time is, by construction, null (snap-stabilization [7]), the system cannot determine whether it is in a convergence phase or if the specifications are currently satisfied. As a consequence, self-stabilizing algorithms are mainly useful when the convergence time is much smaller than the delay between two transient faults. Moreover, for critic applications (those where human security is involved for instance) that need to face the transient faults, other paradigms may be chosen, such as redundancy schemes. This shows a limitation of self-stabilization to build robust distributed systems.

Nevertheless, redundancy schemes are not always possible. They generally rely on a consensus to maintain replication. However, it is well known that it cannot be solved in an asynchronous error-prone system [20]. Moreover, in a dynamic network, the control overhead of the redundancy protocols could be too important. To circumvent the impossibility result, a solution consists in using unreliable failure detector [10]. However their implementations are not straightforward [2] and, as for replication schemes, the control overhead is not really compatible with dynamic networks.

It follows from the above discussions that, when the dynamic increases, 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. We borrow the term "best-effort" from the networking community to qualify these distributed systems. A best effort distributed system fulfills its specifications if the dynamic of the network allows it, and fulfill them few time after the network allows it, otherwise.

In such settings, self-stabilizing algorithms appear then to be a kind of unavoidable approach for designing such best-effort distributed protocols. However, when using the self-stabilization paradigm to tackle the dynamic in ad hoc networks, it is implicitly assumed that the convergence time is smaller than the delay between two topology changes. If it is not the case, the system could never reach a correct state and ensure a correct behavior (with respect to the specifications of the protocol it is running). Self-stabilization property needs then to be completed. This has been studied in [21, 15, 23, 30]. In this paper, we propose another way by adding the notion of *continuity* to the self-stabilization.

Continuity in dynamic networks. Roughly speaking, a highly dynamic distributed system is a system in which the distributed protocol cannot achieve its aim before a new topology change occurs. We introduce the continuity property to complement the self-stabilization in order to build best effort algorithms in dynamic distributed systems.

While the system is converging to a correct behavior, the continuity property ensures that it does not run away, except if it is forced by a topology change.

Indeed, the self-stabilization generally ensures the convergence of the system, providing a stable topology. Moreover, it admits complete reorganization or reset of the whole system whenever a topology change occurs. In a mobile dynamic ad hoc network, if the topology changes occur often, it is possible that the system converges rarely, even never. This means that the service rendered by the distributed protocol will certainly considered (and used) before the convergence. It is then important to ensure some guarantees along the execution: the result should be "better and better", except if a topology change prevents it (but it is expected that not all the topology changes will prevent it). In many cases, this is more useful than to guarantee an asymptotic convergence or a convergence after the dynamic decreases.

Note that our aim is very close to the one introduced in [15]. 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. The continuity property is intended to be fulfilled even 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). This is important in dynamic networks because the frequency of topology changes may delay the convergence to a legitimate configuration. Moreover, to the contrary of the passage predicate, the continuity property is defined on an n-tuple of successive configurations, to indicate that any discontinuity in the successive results is forbidden.

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

Dynamic group 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), which exhibits characteristics that are dramatically different from many generic MANETs [4].

Many VANET applications require a 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, which limits the number of hops, etc. When the group's 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 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 (eg. less groups, no alone 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 succeed in merging. It is then more important to maintain existing groups as long as possible.

To achieve this best effort group services in VANET, we propose a self-stabilizing distributed algorithm in asynchronous unreliable message passing. Each node owns a *local view* of its group; the algorithm stabilizes these views in such a way that all the members of group will eventually share the same view, in which all the members and only them appear. In each node, the local view is used by applications (such as distributed perception or chats). Moreover, our algorithm admits the following continuity property: the view's size of any node never decreases except if a topology change leads to the violation of the diameter constraint (or a node leaves).

Several works deal with the distributed group algorithms in the literature. In static systems, the membership of a node to a group does not change over the time (except if this node want to leaver the group). In dynamic systems, since the nodes are not static, a key component of such structure is the *group membership* service, which is in charge of adding and removing nodes from the group along the time [26]. The group membership service must provide to each node the *view* of the group to which it belongs to. One important feature is that all the nodes belonging to the same group must agree on the same view. In the context of dynamic networks, such agreement on the group view is unsolvable [9]. Dynamic group membership maintenance is discussed in [26] and an algorithm is proposed; it relies on a dynamic atomic broadcast (atomic multicast) protocol provided in the same paper, and assumes failure detectors. In [5], the difficulty to achieve dynamic group maintenance is circumvented by reducing the group membership service to the local environment of a node. This protocol provides the local view of each node but the view of two neighbors could differs. 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 [12], the authors propose a self-stabilizing k-clustering algorithm for static networks. In [16], 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 a group partition limited to k hops.

Group communication structures have been proposed in the literature to achieve fault-tolerance in distributed systems [3], by providing for instance replication, virtual synchrony, reliable broadcast, or atomic broadcast (*e.g.*, [27, 22]). Other works deal with the k-clustering or k-dominating set problem [11, 1, 24, 25] 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. 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. It is designed for unreliable message passing systems on dynamic ad hoc networks. It is self-stabilizing and satisfies a continuity property, ensuring that an application relying on groups could run before convergence of the group service with limited damage.

Contributions. First, we propose a model (Section 2) that take into account the dynamic by defining the *k-dynamic system* (in a sens, this definition makes a link between the nodes speed and the network speed), the k-dynamic self-stabilization and the continuity. Second we specify a new group service (inspired from VANET) along with a continuity property (Section 3). Third we give a distributed protocol to solve this dynamic group service (Section 4). Finally, we prove that the protocol is self-stabilizing in a dynamic system, and fulfills its specifications (Section 5).

2 Model

We consider a system $\mathcal{S}$ composed of mobiles nodes that communicate by wireless communication devices. To fix the ideas, the reader can consider a set of vehicules able to communicate by local broadcast, using IEEE 802.11 devices in ad hoc mode and link layers' protocols.

Nodes. Let V be a set of nodes spread out in an Euclidean space. The total number of nodes in V is finite but unknown. A 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 is either *active* or *passive*. If it is active, a node u can compute, send and receive messages by executing a local algorithm. Nodes are not synchronized. A *distributed* protocol $\mathcal{P}$ is composed of all the local algorithms.

For the needs of our study, we consider a global clock (but nodes do not have access to such a global clock, the system remains asynchronous). A *time instant* t (or *date*) given by this clock is a positive real: $t \in \mathbb{R}^+$. The accuracy of the clock is supposed to be enough compared to speed of the nodes and the communication protocol.

Transmission. A node u can communicate with a node v if $dist(u, v) < r$, where r is the *range* and $dist(u, v)$ denotes the Euclidean distance between u and v. The range r depends on the communication devices. We define the *vicinity* of a node v as the part of the Euclidean space from where a node can receive a message: the node v can receive a message sent by u only if u is in the vicinity of v. The vicinity of v is included in the sphere centered in v and of radius r. Note that a vicinity can be smaller than the sphere of radius r, due to obstacles, power sending, or antenna characteristics. Also, u can be in the vicinity of v while v is not in the vicinity of u (eg. different sending power or antenna).

Node v receives the message m sent by u only if there is no other node than u in its vicinity which is currently sending a message. In the opposite case, the node v will not be able to understand the message (collision). Moreover, v cannot receive a message if it is currently sending a message. Thus, when an active node u sends a message at a time instant t, it can be received by a large set of active nodes into a sphere of range r centered on u. However, a node v can receive a piece of data (that is, any understandable information) from u if (i) both u and v are active, (ii) u is into 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). In order to receive a complete message, these conditions must be satisfied from the begining of the sending by u to the end of the reception by v.

We do not suppose any fair medium access protocol. A node may fail in sending a message because close nodes are continuously sending. Similarly, a node may fail in receiving a message. A node that continuously fail in sending a message is considered as a passive node.

Communication link. 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. Note that, while a communication link exists from u to v at date t, a communication may fail because conditions (iv) and (v) above are not fulfilled, or because the duration of the link is too short. A communication link is bounded.

Topology change. Since nodes can move and change their states, the topology of the system $\mathcal{S}$ evolves in the time. The topology can be affected by a node that disappears or appears, change its state (either active or passive) or moves (leading to some communication link breaks and creation). However, for a given node, all these effects appear in the same way: a change in its neighborhood. Hence, a neighborhood change is a common, node-centered characterization of the topology change, which is more adapted to the design of distributed algorithms.

However, a node will perceive a change either by receiving a message from a new neighbor of by a timer expiration, which, for instance, indicates that an old neighbor seems to have left the neighborhood because it did not send a message recently. Since a node cannot send a message as soon as it want to do it, such timers are generally calibrated with a larger value than the delay required to send a message. Hence, the minimal unit of time is the delay required to send a single message. Any topology change shorter than this unit cannot be noticed. In the following, by *topology change* we consider a change that can be detected by at least one neighbor.

Configuration. A *configuration* c_t of $\mathcal{S}$ is the union of (i) the states of memories of all the processors at time t and (ii) the contents of all the communication links at t. 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 appearance and disappearance of the communication links in the system because of the nodes' moves).

Let $\mathcal{C}$ be the set of configurations of $\mathcal{S}$. Starting from the date t_0, an *execution* of a distributed protocol $\mathcal{P}$ over $\mathcal{S}$ is a sequence of configurations $c_{t_0}, c_{t_1}, \ldots$ 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 can notice the change ($\forall i \in \mathbb{N}, \forall t \in \mathbb{R}^+$, $t_i \leq t < t_{i+1} \Rightarrow c_t = c_{t_i}$) 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 (as defined above: detected by at least one node) leads to a new configuration.

For sake of simplicity, in the following the successive configurations of an execution will be numbered by integers: by stating $c_i = c_{t_i}$, the execution $c_{t_0}, c_{t_1}, c_{t_2} \ldots$ is denoted by $e_{c_0} = c_0, c_1, c_2 \ldots$.

Dynamic system. The *topology* of the distributed system $\mathcal{S}$ at time t is the oriented graph $G_t(V_t, E_t)$ defined as follows: $V_t \subset V$ is the set of active nodes in $\mathcal{S}$ at date t, and $E_t \subset V_t \times V_t$ is the set of communication links at date t.

By definition of the configuration, there is a single topology per configuration; we then denote by G_i the topology of $\mathcal{S}$ during the i-th configuration. Conversely, any topology change leads to a new 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*. Given an execution $e_{c_0} = c_0, c_1, \ldots$ of a dynamic system $\mathcal{S}$, a *topological round* is a maximal sequence of successive configurations $c_i, c_{i+1}, \ldots, c_{i+k}$ during which the topology of $\mathcal{S}$ remains unchanged, *i.e*, $\forall j \in \{i, \ldots, i+k\}$, $G_i = G_j$.

The system $\mathcal{S}$ is *δ-dynamic* if any node u that experiments a neighborhood change is able to diffuse a message to all its neighbors at distance smaller than or equal to δ before the end of the current topological round. More formally, let c_p (resp. c_r) be the first (resp. last) configuration of the kth topological round. Let denote by G_p the directed graph modeling the topology of $\mathcal{S}$ in this topological round, and $d_{G_p}(u, v)$ the distance from nodes u to v in the digraph G_p. Let u be a node that experienced a topology change in c_p, that is, its neighborhood's knowledge in configuration c_p is different than in the previous configuration c_{p-1}. If u diffuses a message in configuration c_p and if other nodes relay this message, then any node v satisfying $d_{G_p}(u, v) \leq \delta$ and active during all the kth topological round will receive the message in a configuration c_q reached before c_r ($p \leq q \leq r$).

Note that a δ-dynamic system is also δ'-dynamic for any δ' satisfying $1 \leq \delta' \leq \delta$. Note also that if a system is not 1-dynamic, then a node could enter in the neighborhood of another one while this last were not able to send a message to all its neighbors. This means that, it is not guaranteed that the two nodes will succeed in exchanging data.

Self-Stabilization. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be subsets of $\mathcal{C}$. Then $\mathcal{C}_2$ is a *closed attractor* for $\mathcal{C}_1$ if and only if for any configuration $c_1 \in \mathcal{C}_1$, and for any execution $e = c_1, c_2, \ldots$, there exists $i \geq 1$ such that for any $j \geq i$, c_j is in $\mathcal{C}_2$.

Define a *specification* of a task as the predicate Π on the set $\mathcal{C}$ of configurations of system $\mathcal{S}$. A *legitimate configuration* for this task is a configuration $c \in \mathcal{C}$ satisfying $\Pi(c)$. We denote by $\mathcal{L} \subset \mathcal{C}$ the set of all legitimate configurations for this task. Let $\mathcal{P}$ be a distributed protocol solving this task in the system $\mathcal{S}$. Then $\mathcal{P}$ is self-stabilizing in $\mathcal{S}$ if $\mathcal{L}$ is a close attractor for $\mathcal{C}$.

A distributed protocol is *δ-dynamic self-stabilizing* with respect to a predicate Π, if it is self-stabilizing for Π in a δ-dynamic distributed system. The more δ is small, the more the protocol is able to stabilize while the dynamic of the system is high.

Continuity property. The properties of a distributed protocol can be characterized by properties on the successive configurations of the distributed system running it.

A *safety property* Π_S ensures that any reached configuration admits some properties. It is generally used to indicate that nothing bad will happen. We introduce the *continuity property* (denoted Π_C), which ensures that any new configuration takes into account the previous ones. It is used to indicate that a new result cannot be less interesting than the previous one. A generic definition of a continuity property is: for any execution e of the system running the protocol, there exists $l > 0$ such that, for any configuration $c_i \in e$, $\Pi_C(c_i, \dots, c_{i+l})$ holds.

3 Dynamic Group Service

In this section, we specify a new group service for dynamic networks. The motivation comes from vehicular ad hoc networks.

3.1 Informal Description and Motivation

Context. Let consider a vehicular ad hoc network, and assume that, in each vehicle, a passenger want to participate in a distributed application (such as a chat, or a distributed game), called in the following (*distributed*) *application*. Such an application cannot run correctly if the communication delay between two participants is too high. Hence, not all the vehicles can play together and several instances of the distributed application run in the vehicular network. The passenger of a given vehicle cannot participate simultaneously to several instances of the application. There is a single passenger per vehicle and we indifferently use vehicle, passenger, participant or node.

Group. The vehicles that play to the same instance of the application form a *group*. The diameter of each group is bounded by a given constant, in order to fulfill the application constraint (assuming here that the communication delay is proportional to the number of hops). Moreover, groups are disjoint.

When a message related to the distributed application is sent by a vehicle, it will be relayed by the nodes of its group until all vehicles belonging to the group received it. How to perform such a message distribution inside the group is out of the scope of this paper. However, a vehicle will not relay a message sent by a node that does not belong to its group, in order to limit messages propagation and to reduce bandwidth consumption. As a consequence, the groups must be connected, meaning that, for any pair of nodes u, v in a given group, there exists a path from u to v composed with only nodes of the group.

A group always accept a new participant (passenger), providing that the diameter constraint is still fulfilled when it is accepted.

Continuity. At any time, depending on the vehicles' moves, the topology of the network is changing. As a consequence, the diameter of a given group could be larger than the constant fixed by the distributed application. In this case, some passengers have to leave the group to satisfy the diameter constraint.

However, in order to offer the better service to the passengers, the composition of the group should be modified if and only if the vehicles' moves modify the group diameter. Hence, if some vehicles participate to the same instance of the distributed application, the group service should maintain as long as possible the group. Thus, the problem differs from any optimal overlapping of the network topology by cliques. Indeed, let us consider a network as a convoy of vehicles $a-b-c-d-e$ and suppose that the maximal admissible diameter is two. From a global point of view, an optimal group organization would be for instance (a,b,c) and (d,e). However, if the passengers of vehicles b, c and d began to play together, the service should prevent any unnecessary split of the group (b,c,d) in order to ensure a durable service to these passengers.

This constraint is justified by the fact that the topology of a vehicular network is highly dynamic. The existing groups should then not be challenged by the frequent changes in the network: the arrival of vehicle a at the end of convoy $b-c-d$ should not interrupt existing instance of the application. Second, it is illusory to seek any global topological optimization in such a dynamic network because its duration will be very short. Third, since the topology is dynamic, the nodes a and e (which were alone in the previous example) will not remain alone during a long time. For instance, if vehicle a moves close to c, it will be able to join the group (b,c,d) without changing its diameter.

3.2 Specification

Group organization. This kind of group service builds an organization of the nodes. Let `Dmax` be an integer representing the maximal admissible diameter for a group. We define a *group organization* of the nodes, denoted Ω, as follows:

- Ω is a partition of the set of vertices V:
  $\exists p \in \mathbb{N}$ such that $\Omega = \{\Omega_1, \Omega_2, \ldots, \Omega_p\}$ with $\Omega_i \subset V$ for all $i \in \{1,\ldots,p\}$
  $\forall i,j \in \{1,\ldots,p\}$, $\Omega_i \cap \Omega_j = \emptyset$ and $\bigcup_{i=1}^{p} \Omega_i = V$.
- Each Ω_i is connected and satisfies the constraint on the diameter:
  $\forall \Omega_i \in \Omega$, $\forall u,v \in \Omega_i$, $0 < d_{\Omega_i}(u,v) \leq \mathtt{Dmax}$
  where d_{Ω_i} is the distance in the subgraph composed of vertices of Ω_i.

In the following, a set $\Omega_i \in \Omega$ is called a *group*. We denote by $\Omega(c)$ the group organization in configuration $c \in \mathcal{C}$. A group organization is *maximal* if by merging two of its groups, we cannot obtain a correct group organization: $\forall \Omega_i, \Omega_j \in \Omega$, $\Omega \setminus \{\Omega_i, \Omega_j\} \cup \{\Omega_i \cup \Omega_j\}$ is not a correct group organization. For instance, in the convoy of vehicles $a-b-c-d-e$ with $\mathtt{Dmax} = 2$, the group organization $\Omega = \{\{a\},\{b,c,d\},\{e\}\}$ is maximal because $\Omega \setminus \{\{a\},\{b,c,d\}\} \cup \{\{a\} \cup \{b,c,d\}\} = \{\{a,b,c,d\},\{e\}\}$ is not a correct group organization (the diameter of the group $\{a,b,c,d\}$ is larger than `Dmax`) and the same holds with other possible merging.

Dynamic group services protocol. The problem statement consists in designing a distributed protocol that gives, for each node, the composition of its group. This protocol is called *Dynamic group protocol*. We suppose that each node v owns a variable $\mathtt{view}_v$, which stores its current knowledge of its group. The application (distributed game, chat...) running on the node v will consider the nodes of view_v as the current members of the group of v. We denote by $\mathtt{view}_v(c)$ the value of $\mathtt{view}_v$ in configuration c.

A *legitimate configuration* is a configuration c in which the variables $\mathtt{view}_v(c)$, $v \in V$, define a maximal group organization. Let Π_A be a predicate defined on the configurations and called *agreement property*: $P_A(c)$ holds if $\forall v \in V$, $\forall w \in \texttt{view}_v(c)$, $\texttt{view}_v(c) = \texttt{view}_w(c)$. Let Π_S be a predicate defined on the configuration and called *safety property*: $\Pi_S(c)$ holds if the diameter of each group in the configuration c is smaller than `Dmax`.

Before a legitimate configuration is reached, the distributed application will use information given by the dynamic group protocol. This protocol should then fulfill some properties to give some guarantees on its outputs. We define a *continuity property* Π_C on the two-tuples of configurations in order to ensure that the successive outputs given to the application are coherent: a node v leaves its group only if it is necessary to satisfy the safety property after a topology change. More formally, for two successive configurations $c_i, c_{i+1} \in \mathcal{C}$, $\Pi_C(c_i, c_{i+1})$ is true if

$$\forall v \in V, \texttt{view}_v(c_i) \not\subseteq \texttt{view}_v(c_{i+1}) \Rightarrow \exists w \in \texttt{view}_v(c_i), d_{G_i}(v, w) > \texttt{Dmax}$$

where d_{G_i} denotes the distance in the topology of the system in configuration c_i.

Remark. Note that this problem is different from graph covering, maximal independent set, or dominating set problems [24]. Moreover it differs from the k-clustering problem [12]. To the best of our knowledge, the continuity constraint leads to an original problem.

4 Distributed protocol

4.1 Principe

For a given node, the candidates to form a group are neighbors up to distance `Dmax`. Nodes build such lists by diffusing messages in their neighborhood. Only symmetric links are taken into account. In $O(\texttt{Dmax})$ the knowledge of the `Dmax` neighborhood can be known. However, when a node receives a list which is too long compared to its current list, it will reject it to avoid any split of its current group. Moreover, bad lists are rejected (such as lists larger than `Dmax`). When a node enters in a new group, its arrival will be propagated in $O(\texttt{Dmax})$ to the group's members. Such an arrival can increase the diameter of the group. A new member will be accepted only if the diameter constraint is respected. In some cases, two nodes can be accepted by two distant group's members and the diameter constraint is no more fulfilled. In this case, one of the new member must leave the group (instead of splitting the existing group). To chose which of them leaves the group, the protocol uses the lexicographical order on the nodes' identities.

Truncated lists of neighbors are built using a simple distributed algorithm relying on an *r-operator* [17]. This operator computes the complete ordered list of ancestors' sets [19]. Thanks to its properties, the protocol is self-stabilizing [13]. The rest of the dynamic group protocol consists in checking whether a list can be accepted or not in order to fulfill the safety property Π_S and the continuity property Π_C. Our protocol being self-stabilizing, the agreement property will eventually be fulfilled. Even if the system has not converged, the applications (chat, distributed perception...) will use the view built by the dynamic group protocol (best effort approach in dynamic system). The continuity property ensures that the view cannot be reduced meaning that a collaborative work which has started will not stop, except if the safety property would not be respected because of the dynamic of the network.

4.2 Building the lists of ancestors' sets

Let denote by $\vec{d}(u, v)$ the distance from u to v in the oriented graph $G(V, E)$ (minimal number of edges to reach v from u). We denote by $d_v = \max_{u \in V} \vec{d}(u, v)$ the largest distance from any vertex of V to v. We denote by $a_v^i = \{u \in V, \vec{d}(u, v) = i\}$ the set of vertices (ancestors) which are at distance i of v. We state $a_v^0 = \{v\}$. The *list of ancestors' sets* of a node v is defined by: $(a_v^0,\ a_v^1,\ \ldots,\ a_v^{d_v})$. A list of ancestors' sets is *partial* if some vertices are lacking in some ancestors' sets. A list of ancestors' sets is *truncated* if some sets are lacking at the end of the list. Note however that a list of ancestors' sets of a given node v always admit $\{v\}$ as first set.

When modeling distributed algorithms with algebraic operators, interesting properties (termination, self-stabilization) can be ensured by simply checking some local properties of the operator. To build the list of ancestors' sets, we use the r-operator `ant` [17, 19, 13]. We consider the set $\mathbb{S}$ composed of the 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 $\mathbb{S}$. We define on $\mathbb{S}$ the operator $\oplus$ 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, we define the endomorphism r 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 $\triangleleft$ by: $l_1 \triangleleft l_2 = l_1 \oplus r(l_2)$, where l_1 and l_2 are lists from $\mathbb{S}$. This is a strictly idempotent r-operator [17] called `ant`, that induces a partial order relation. It leads to self-stabilizing static tasks (building the complete ordered lists of ancestors' sets) in the register model [19]. Since our communication model (defined for IEEE 802.11 networks) admits bounded links, these results can be extended to this kind of message passing model. (Refer to the discussion related to r-operators in wireless networks in [13].)

4.3 Algorithm

The distributed protocol *Dynamic Group Service* is composed of a single algorithm per node (uniform protocol). This algorithm uses a timer to regularly send a message in the neighborhood. All messages received from the neighborhood are collected. If a neighbor sends more than one message before the timer expiration, only the last received is kept. If no message has been received from a neighbor, it disappears from the list of the neighbors. Each time the timer expires, the node computes the truncated list of ancestors' sets by using those received from its neighbors in the messages. This list will be sent in the neighborhood. It also updates the view of its own group (that is, its knowledge of the composition of its group). The view is the output of the protocol; it is used by the application (eg. chat, collaborative perception...) which requested the dynamic group service, and which gave the diameter constraint (which is fixed during all the execution).

The algorithm uses the following variables: `msgSet` contains the last received messages, `nbhSet` contains the identities of the known neighbors, `lstAnt` contains the truncated list of ancestors, `view` is the output, `Dmax` is a constant integer given by the application (and regularly set to avoid any fault on this memory), and `period` is the timer duration. Since the convergence time is at most `Dmax` timers (Section 5), only nodes which are present in the list of ancestors' sets from `Dmax` timers are taken into account to build the view. This allows to avoid disturbing the application during the convergence phase, or when a neighbor tries to join the group until it has been accepted.

The notation `msgSet` $_{|u}$ denotes the message sent by u and stored in the set `msgSet`. Since such a message is a list of ancestors' sets, the notation `msgSet` $_{|u}.i$ denotes the ith set of ancestors in the list. The size of a list l is denoted by $s(l)$. At the reception of a message, the sender u is known by the receiver v thanks to the first set of the received list $\texttt{msgSet}_{|u}$ ($\texttt{msgSet}_{|u}.0 = \{u\}$). If v does not belong to the neighbors of u ($v \notin \texttt{msgSet}_{|u}.1$), it suspects an asymmetric link and then marks the sender in its list of ancestors' sets (denoted by $\underline{u}$). When the link becomes symmetric (u has also detected v), either u and v belongs to the same group or v deduced that it cannot belong to the same group of u. In the first case, u is no more marked in the list of v. In the second case, it is double-marked (denoted by $\underline{\underline{u}}$). The algorithm is given below.

Algorithm *Dynamic Group Service* (node v)

```
1   Upon reception of a message msg sent by a node u:
2     nbhSet ← nbhSet ∪ {u}                              ▷ updating the set of neighbors
3     msgSet ← msgSet \ {msgSet|u} ∪ {msg}               ▷ updating msgSet|u
4   Upon timer expiration:
5     compute-lstAnt()                                    ▷ updating lstAnt
6     send( lstAnt ) to the neighbors
7     view ← non marked nodes which are in lstAnt from Dmax timers
8     msgSet ← empty list                                 ▷ reset of the neighborhood data
9     nbhSet ← empty list
10    restart timer with duration period
```

The procedure compute-lstAnt() builds the truncated list of ancestors, that will be sent in the neighborhood. The computation is based on the ant r-operator applied on incoming lists received by the neighbors. However, not all the lists are taken into account, in order to fulfill the safety and continuity properties.

First the marked nodes, which indicates asymmetric links, are only admitted in the neighborhood (Line 2). This prevents any propagation of asymmetric link information. Next, only lists which are coherent are accepted (Line 4). In the converse case, they are replaced by a list containing only the marked sender, meaning that only the information regarding the sender is kept. A bad list is a list that does not contain v nor $\underline{v}$ in place 1, or a list which is too long, or a list that contains an empty set among the sets of ancestors. Next, only lists which are compatible with the last computed list is kept (Line 6). An incompatible list is a list sent by a node which was not already in the last computed list of ancestors' sets and which does not fulfill the technical condition of Lemma 1. This condition ensures that, by taken into account the list sent by a new neighbor, the node will not split its current group. This is important for the continuity property. A neighbor that sent an incompatible list is double-marked ($\underline{\underline{u}}$) in order to notify that it is not accepted.

After the purges in the incoming lists, a first computation of the list of ancestors' sets is performed, using the ant r-operator (Lines 10-13). Note that this computation ought to be performed inside the first *forall* loop but we preferred to separate it for clarity.

While some purges have been done in the incoming lists, the computed list of ancestors could reach the size of $\texttt{Dmax} + 2$ while the maximum is $\texttt{Dmax} + 1$. 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 lexicographical order on the identities of the nodes (Line 16). If the local node v is not smaller than a given too far node w, the list in which w appears is ignored (Line 19). At the opposite side of the group, node w keeps the list containing v but the end of its ancestors' list will be truncated (meaning that v and w will not belong to the same group). Indeed, after all the too far nodes have been examined, the list of ancestors is computed again (Lines 24-27) and is truncated (Line 28) in order to delete the too far nodes having less priority.

Procedure compute-lstAnt()

```
▷ Purging the last received lists of ancestors' sets.
for all u ∈ nbhSet do
   delete marked nodes except v̲ in msgSet|u
   if badList(msgSet|u) then
      msgSet ← msgSet \ msgSet|u ∪ (u̲)
   end if
   if incompatibleList(msgSet|u) then
      msgSet ← msgSet \ msgSet|u ∪ (u̲̲)
   end if
end for

▷ Computing the list of ancestors' sets.
lstAnt ← (v)
for all u ∈ nbhSet do
   lstAnt ← ant(lstAnt, msgSet|u)
end for

▷ Removing incoming lists containing too far nodes with priority.
if s(lstAnt) = Dmax + 1 then                      ▷ the list is too long
   for all w ∈ lstAnt.Dmax do                     ▷ scanning the last place of the list
      if w < v then                               ▷ the node has not the priority.
         for all u ∈ nbhSet do                    ▷ looking for lists that provided w.
            if w ∈ msgSet|u.(Dmax − 1) then
               ▷ the neighbor that provided w is ignored.
              msgSet ← msgSet \ msgSet|u ∪ (u̲̲)
            end if
         end for
      end if
   end for

   ▷ Computing the list again and truncate too far node with less priority.
   lstAnt ← (v)
   for all u ∈ NbhSet do
      lstAnt ← ant(lstAnt, MsgSet|u)
   end for
   lstAnt ← (lstAnt.1, lstAnt.2 ..., lstAnt.(min(s(lstAnt), Dmax))
end if
```

Procedure badList(list)

```
if v and v̲ are not in list.1 or s(list) > Dmax or ∅ ∈ list then
   return true
end if
```

Procedure incompatibleList(list)

```
if list.1 ∉ lstAnt then                           ▷ the sender is a new neighbor.
   if s(lstAnt) + s(list) > Dmax and ∀i ∈ {0, ..., s(lstAnt)},
    lstAnt.i ⊄ list.1 or
    min (s(lstAnt) + s(list) + 1 − i, s(list) + 1 + i/2) > Dmax then
      return true                                 ▷ see Lemma 1.
   end if
end if
```

5 Sketch of proof

To prove the self-stabilizing property of the algorithm, we prove that it converges in finite state to a maximal group organization after the last transient fault or the last topology change. In other words, it converges to a legitimate configuration starting from *any* configuration of $\mathcal{C}$. Then we prove that the algorithm satisfies the continuous property, which ensure that the groups' composition is only adapted when necessary. In other words, the arrival of a new vehicle on the vicinity of a group does not challenge the group composition.

We first prove that incorrect values leave in finite time the network. Thanks to the badList test, malformed lists disappear after a timer expiration. However a list could still contain bad identities. When a node sends a list, it will be either ignored by its neighbors or taken into account. When it is taken into account, all the elements will be shift on the right by the ant computation. After k hops, the list has been shift by k places to the right. Since the lists are truncated, a node never sends a list larger than `Dmax`. Hence, starting from any configuration, the system reaches a configuration after k timer expirations where all the nodes contain in their `msgSet` correct partial lists of ancestors' sets. Let denote by $\mathcal{L}' \subset \mathcal{C}$ the set of all such configurations; $\mathcal{L}'$ is a close attractor for $\mathcal{C}$.

We now prove that, starting from a configuration of $\mathcal{L}'$, the system will stabilize on a legitimate configuration of $\mathcal{L}$, that is, a configuration where the views on each node define a maximal group organization.

Let us denote by $\mathrm{partners}_v(c)$, the set of non-marked nodes belonging to the variable `lstAnt` on node v in the configuration c. We then define the set of nodes $\mathrm{group}_v(c)$ as follows: if, for each node $w \in \mathrm{partners}_v(c)$, $\mathrm{partners}_w(c) = \mathrm{partners}_v(c)$ then $\mathrm{group}_v(c) = \mathrm{partners}_v(c)$, else $\mathrm{group}_v(c) = \{v\}$. Hence, $\mathrm{group}_v(c)$ denotes the valid group to which node v belongs in configuration $c \in \mathcal{L}'$. Denote by $\mathrm{ng}(c)$, the number of different groups in the configuration c: $\mathrm{ng}(c) = |\{\mathrm{group}_v(c), v \in V\}|$. We prove that $\mathrm{ng}(c)$ decreases until reaching a minimum. We call *external edge* an edge $(u, v) \in E$ which connects two nodes that do not share the same group in the current configuration c: $\mathrm{group}_u(c) \neq \mathrm{group}_v(c)$. Whenever an external edge disappears, the number of groups decreases (two groups merged). Whenever the number of groups increases, the number of external edges increases too.

We prove that no external edges is created starting from a configuration of $\mathcal{L}'$. There are three cases in the algorithm where a node v does not accept a neighbor u (*i.e.*, u is double-marked). The first case is when u sends a bad list. This case cannot happen starting from a configuration in $\mathcal{L}'$. The second case occurs when u sends an incompatible list, which can happen only if u was not already in the group of v. Thus, an external link is not created between u and v. The third case happens when u knows a far neighbor w such that both v and w cannot belong to the same group and w has the priority to v (Line 19). Symmetrically, w will ignore the far neighbor v by truncating its list (Line 28). However, in this situation, w was not in $\mathtt{lstAnt}_v$ while it was in $\mathtt{lstAnt}_u$ and then u and v do not belong to the same group. Thus, an external edge is not created in the third case. Hence, the algorithm does not create external edges starting from configurations of $\mathcal{L}'$.

We prove that the number of external edges decreases when starting from configurations of $\mathcal{L}' \setminus \mathcal{L}$. Let us consider a symmetric edge (u, v) (asymmetric edges are ignored by the algorithm, thanks to nodes' marking). Since the edge is symmetric, either u or v will eventually receive a correct partial list of ancestors from the other extremity of the edge. Let u be the node that first receive a list from the other extremity. This list does not contain u, and the node u will then broadcast a list containing $\underline{v}$. When v receives such a list, it discovers that (u, v) is a symmetric edge and takes into account the list sent by u. Three cases are possible. First, the list sent by u could be incompatible with v's. In this case, v sends a list containing $\underline{\underline{u}}$. Second, the list of u is compatible but the new list computed by v is too long due to a far node w such that $w < v$. In this case, the node v will ignore the list sent by u; it sends a list containing $\underline{\underline{u}}$. Third, the list of u is compatible and it does not provide any too far node w with $w < v$. In this case, the node v will send a list that contains u. The node u will eventually receive the list of v. Again, the three cases already described can apply and u will either send a list with $\underline{\underline{v}}$ or with v.

Now, let consider the two cases (either receiving $\underline{\underline{v}}$ or v). Whenever u or v sends a list with the other extremity double-marked, this means that u and v cannot belong to the same group. If neither u nor v sent a list with the other extremity double-marked, then u and v will eventually appear in the same group. However, in case of one of them (say v) truncated its list to avoid a too far node w with less priority than it ($v < w$), then u and v have not the same list: w still belongs to the list of u while it does not appear in the list of v). After at most `Dmax` timer expirations, w will receive a list containing v. Since v is too far from w and since $v < w$, w will ignore the neighbors that sent a list containing w. After at most `Dmax` timer expirations, u will receive lists that does not contain w since no more neighbor of w will add w in their list. Thus w will disappear in at most `Dmax` timer expirations from the list of u; u and v appear then in the same group. So, considering an external edge (u, v), either it always remain external (one of the extremity double-marked the other), or it becomes an internal edge in less than `Dmax` timer expirations.

Finally, when no external edge can disappear, this means that no pair of groups can merge while satisfying the safety property Π_S. As a consequence, a maximal group organization has been reached. Hence, starting from a configuration of $\mathcal{L}'$, a configuration of $\mathcal{L}$ is reached. Moreover, since the algorithm does not create external edges, it does not split existing groups after the last transient fault or topology change. Hence, starting from a configuration of $\mathcal{L}$, the system cannot reach a configuration outside $\mathcal{L}$. Then $\mathcal{L}$ is a close attractor for $\mathcal{C}$ and the protocol is self-stabilizing in $\mathcal{S}$ (it is `Dmax`-dynamic self-stabilizing).

It remains to prove the continuity property Π_C. Suppose that, after a node v accepts the list of a new neighbor, its group is split. Then, by Lemma 1, this means that v accepted an incompatible list, which is impossible (Line 6). Note that if a node would have a bad list (containing a too far or inexistent node), then the distance to this node is larger than k, which contradicts the safety property and the split is allowed.

Lemma 1 *Let v and w be two nodes owning the partial truncated lists of ancestors' sets (a_v^0, a_v^1, $\ldots$, a_v^p) and ($a_w^0, a_w^1, \ldots, a_w^q$) respectively, and suppose that w arrives in the vicinity of the group of v. Then v can accept the list of w without challenging the current groups 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 $p-i+1+q \leq \mathtt{Dmax}$ or $i/2+q+1 \leq \mathtt{Dmax}$.*

6 Conclusion

This paper introduces the continuity properties to complete the self-stabilization in dynamic ad hoc networks. A group service for these networks is specified, and a distributed protocol has been designed to solve it. This protocol maintains the groups' diameter smaller that `Dmax`, a bound fixed by the applications' requirements. It can recover from transient faults. It ensures the continuity of the groups' composition, while the nodes's moves do not contradict the diameter constraint. The protocol is implemented; performances are currently studied.

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] M. Bertier, O. Marin, and P. Sens. Implementation and performance evaluation of an adaptable failure detector. In *DSN '02: Proc. of the 2002 Int. Conf. on Dependable Systems and Networks.*

[3] K.P. Birman. The process group approach to reliable distributed computing. *CACM*, 36(12):37–53, 1993.

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

[5] Linda Briesemeister and Günter Hommel. Localized group membership service for ad hoc networks. In *31st Int. Conference on Parallel Processing Workshops (ICPP 2002 Workshops)*, pages 94–102, 2002.

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

[7] Alain Bui, Ajoy Kumar Datta, Franck Petit, and Vincent Villain. Snap-stabilization and PIF in tree networks. *Distributed Computing*, 20(1):3–19, 2007.

[8] B. Bui Wuan, A. Ferrera, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(2):267–285, 2003.

[9] T. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost. On the impossibility of group membership. In *In 15th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 322–330, 1996.

[10] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

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

[12] A. K. Datta, L. L. Larmore, and P. Vemula. A self-stabilizing O(k)-time k-clustering algorithm. *To appear in Computer Journal*, 2008.

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

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

[15] S. Dolev and T. Herman. Superstabilizing protocols for dynamic distributed systems. In *Proc. of the fourteenth annual ACM Symp. on Principles of distributed computing (PODC)*, page 255, 1995.

[16] 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.

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

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

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

[20] M.J. Fischer, N.A. Lynch, and M.S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32:374–382, 1985.

[21] S. Gosh, A. Gupta, and S.V. Pemmaraju. Fault-containing network protocols. In *SAC '97: Proceedings of the 1997 ACM symposium on Applied computing*, pages 431–437, New York, NY, USA, 1997. ACM.

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

[23] S. Kutten and B. Patt-Shamir. Stabilizing time adaptive protocols. *Theor. Comp. Sci.*, 220(1):93–111, 1999.

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

[25] L. D. Penso and V. C Barbosa. A distributed alg. to find k -dominating sets. *DAM*, 141(1-3):243–253, 2004.

[26] A. Schiper. Dynamic group communication. *Distributed Computing*, 18(5):359–374, 2006.

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

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

[29] George Varghese and Mahesh Jayaram. The fault span of crash failures. *JACM*, 47(2):244–293, 2000.

[30] Hongwei Zhang and Anish Arora. Guaranteed fault containment and local stabilization in routing. *Computer Networks (Elsevier)*, 50(18):3585–3607, to appear.

A Proof of Lemma 1

Let w be the first node of group$_w$ for which the list of ancestors' set is received by v. Then, the only external edges between group$_v$ and group$_w$ known by v are those joining w. Indeed, the external edges are not propagated in the lists (they involve double-marked nodes). Hence, without loss of generality, we suppose that only these external edges exist between the other groups.

Suppose that the lemma 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 (ie. 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 hypothesis.

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.

Suppose now that the conditions are not fulfilled and that v accepts the list of w. Then the nodes of group$_w$ will be propagated in the lists of nodes of group$_v$ and reciprocally. But at least one node $u \in$ group$_v$ will see that a node $u' \in$ group$_w$ is too far from him and reciprocally. Either u or u' will reject the lists of its neighbors that contain the too far node (depending on the lexicographical order between u and u') and either the group of v or the group of w split. $\square$

B 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 [18]. 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 $x \mapsto x$ [17]. 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 [17] : 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 [13], 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.