Robust Self-Stabilizing Weight-Based Clustering Algorithm

Colette Johnen, Le Huy Nguyen

LRI, Univ. Paris-Sud, CNRS
F91405, Orsay, France

Abstract

Ad hoc networks consist of wireless hosts that communicate with each other in the absence of a fixed infrastructure. Such networks cannot rely on centralized and organized network management. The clustering problem consists of partitioning network nodes into non-overlapping groups called clusters. Clusters give a hierarchical organization to the network that facilitates the network management and that increases its scalability.

In a weight-based clustering algorithm, the clusterheads are selected according to their weight (a node's parameter). The higher the weight of a node, the more suitable this node is for the role of clusterhead. In ad hoc networks, the amount of bandwidth, memory space or battery power of a node could be used to determine weight values.

A self-stabilizing algorithm, regardless of the initial system configuration, converges to legitimate configurations without external intervention. Due to this property, self-stabilizing algorithms tolerate transient faults and they are adaptive to any topology change.

In this paper we present a robust self-stabilizing weight-based clustering algorithm for ad hoc networks. The robustness property guarantees that, starting from an arbitrary configuration, after one asynchronous round, the network is partitioned into clusters. After that, the network stays partitioned during the convergence phase toward a legitimate configuration where the clusters verify the "ad hoc clustering properties".

Key words: Distributed algorithm, Ad hoc networking, Weight-Based Clustering, Self-Stabilization, Robustness.

1 Introduction

An *ad hoc* network is a self-organized network, especially those with wireless or temporary plug-in connections. Such a network may operate in a standalone

Preprint submitted to Elsevier *17 March 2008*

fashion, or may be connected to the larger Internet [14]. In these networks, mobile routers may move arbitrary often; thus, the network's topology may change rapidly and unpredictably. Ad hoc networks cannot rely on centralized and organized network management. Significant examples include establishing survivable, efficient, dynamic communication for emergency/rescue operations, disaster relief efforts, and military networks. Meetings where participants aim at creating a temporary wireless ad hoc network is another typical example. Quick deployment is needed in these situations.

Clustering means partitioning network nodes into groups called clusters, providing the network with a hierarchical organization. A cluster is a connected subgraph of the global network composed of a clusterhead and ordinary nodes. Each node belongs to only one cluster. In addition, a cluster is required to obey certain constraints that are used for network management, routing methods, resource allocation, etc. By dividing the network into non-overlapped clusters, intra-cluster routing is administered by the clusterhead and inter-cluster routing can be achieved in a reactive manner between clusterheads. Thus, clustering-based routing reduces the amount of routing information propagated in the network. Clustering facilitates the reuse of resources, which improves the system capacity. Members of a cluster can share resources such as software, memory space, printer, etc. Moreover, clustering can be used to reduce the amount of information that is used to store the network state. Distant nodes outside of a cluster usually do not need to know the details of specific events occurring inside this cluster. Indeed, an overview of the cluster's state is generally sufficient for those distant nodes to make control decisions. Thus, the clusterhead is typically in charge of collecting the state of nodes in its cluster and constructing an overview of its cluster state.

For the above mentioned reasons, it is not surprising that several distributed clustering algorithms have been proposed during the last ten years [1–3,8,13,15,21]. The clustering algorithms in [1,13] construct a spanning tree. Then the clusters are constructed on top of the spanning tree. The clusterheads set do not necessarily form a dominating set (i.e., a node can be at distance greater than 1 from its clusterhead). Two network architectures for MANET (Mobile Ad hoc Wireless Network) are proposed in [15,21] where nodes are organized into clusters. The clusterheads form an independent set (i.e., clusterheads are not neighbors) and a dominating set. The clusterheads are selected according to the value of their IDs.

In [8], a weight-based distributed clustering algorithm taking into account several parameters (node's degree, transmission and battery power, node mobility) is presented. In a neighborhood, the most suitable for the clusterhead role (i.e., a node optimizing all the parameters) are the selected nodes. In [3], a Distributed and Mobility-Adaptive Clustering algorithm, called DMAC, is presented. The clusterheads are selected according to a node's parameter (called

weight). The higher the weight of a node, the more suitable this node is for the role of clusterhead. An extended version of this algorithm, called Generalized DMAC (GDMAC), is proposed in [2]. In the latter algorithm, the clusterheads do not have to form an independent set. This implies that, when, due to the mobility of the nodes, two or more clusterheads become neighbors, none has to resign. Thus, in highly mobile environment the clustering management with GDMAC requires less overhead than the clustering management with DMAC. The DMAC and GDMAC algorithms are analyzed respectively in following papers [6,7], with respect to their convergence time and message complexity. In [8], a weight-based distributed clustering algorithm is presented; also the computation of the node's weight according to several parameters (node's degree, transmission power, battery power, ...). In [17,25] probabilistic clustering constructions for ad hoc sensor networks are presented.

A system is self-stabilizing when regardless of its initial configuration, it is guaranteed automatically to reach a legitimate configuration in a finite number of steps. The correctness of self-stabilizing algorithms does not depend on initialization of variables, and a self-stabilizing algorithm converges to some predefined stable configuration starting from an arbitrary initial one. Therefore self-stabilizing algorithms are inherently tolerant to transient faults in the system. The self-stabilizing algorithms can also adapt dynamically to changes in the network topology or system parameters (e.g., communication speed, number of nodes). A new configuration resulting from a topological change is viewed as an inconsistent configuration from which the system will converge to a configuration consistent with the new topology. Several self-stabilizing algorithms for cluster formation and clusterhead selection have been proposed [5,11,16,18,22–24]. [16] presents a self-stabilizing algorithm that constructs a maximal independent set (*MIS*) (i.e., members of the set are not neighbors, and the set is maximal to this property). Note that a maximal independent set is a good candidate for the clusterheads set because a maximal independent set is also a dominating set (i.e., any node is member of the dominating set or has a neighbor that is member of the set). In [11], a self-stabilizing algorithm for the construction of wireless connected overlays is presented. Based on the construction of MIS, the authors computed a connected dominating set. In [24], a self-stabilizing algorithm that creates a minimal dominating set (i.e., if any member of the set leaves the set, the set is not further a dominating set) is presented. Note that a minimal dominating set is not necessarily an independent set. In [5], a self-stabilizing link-cluster algorithm under an asynchronous message-passing system model is presented (no convergence proofs are presented). The definition of cluster is not exactly the same as ours: an ordinary node can be at distance two of its clusterhead. The presented clustering algorithm requires three types of messages, our algorithms adapted to message passing model require one type of message. A self-stabilizing algorithm for cluster formation is presented in [23]. A density criterion (defined in [22]) is used to select clusterhead: a node v chooses in its neighborhood the

node having the highest density. A v's neighbor contains all nodes at distance less or equal to 2 from v. Therefore, to choose clusterhead, communication at distance 2 is required. Our algorithms builds clusters on local information; so it requires only communication between nodes at distance 1 of each others. In [10], a probabilistic self-stabilizing clustering algorithm is presented, the clusterheads are randomly selected; in the average a MIS is built in $O(\lg(|V|)$ asynchronous rounds where $|V|$ is the network size.

Both algorithms DMAC and GDMAC are not self-stabilizing, i.e., they work assuming correct initialization. They cannot cope with the wake up problem. Sensors to conserve energy sleep a large portion of the time. During the sleeping period of a sensor, the network topology may have drastically changed. The sensor has to automatically adapt to the new situation. A self-stabilizing version of DMAC and GDMAC is presented in [18]: they cope with any initial configuration. They also adapt to arbitrary topology changes due to node crash failures, communication link crash failures, node recovering or link recovering, merging of several networks, and so on.

In this paper, we present a robust and self-stabilizing version of GDMAC and DMAC. The obtained clusters satisfy the "ad hoc clustering properties", informally presented as follows:

(1) each node is at most at distance 1 from the clusterhead of its cluster.

(2) in a neighborhood there are at most k clusterheads (k being a given parameter).

(3) the clusterhead of a node is nearly the best choice: its clusterhead was a nearly optimal weight (its weight is at most h smaller than the optimal weight).

Starting from an arbitrary configuration, the system satisfies the safety predicate in one synchronous computation step (i.e., one asynchronous round). Once the system satisfies the safety predicate, the system performs correctly its task (i.e., the network is partitioned into clusters). The partition may have to change to get a partition satisfying the ad hoc clustering properties. During the construction of the final clusters the safety predicate stay verified: the network is always partitioned. That is why we call this algorithm robust. The algorithm in [18] is not robust: a node may not belong to a cluster during the stabilization phase even if it belongs initially to a well-formed cluster. In [20] a robust self-stabilizing version of DMAC under synchronous scheduler is presented. Our algorithm is adapted to 1-hop clusters formation algorithms presented in [4,9,12,15].

The stabilization time or convergence time is the time needed to build clusters having the ad hoc clustering properties from any initial configuration, along any computation. The nodes have various speed therefore the convergence time

is established in term of asynchronous rounds. Our algorithm has the following upper bound on the convergence time : $2D+4$ asynchronous rounds, where D is the network diameter. This upper bound is formally proved in Section 7.

Our algorithm is designed for the state model. Nevertheless, it can be easily transformed into an algorithm for the message-passing model. For this purpose, each node v periodically broadcasts to its neighbors a message containing its state. Based on this message, v's neighbors decide whether to update their variables or not. After a change in the value of v's state, node v broadcasts to its neighbors its new state.

The paper is organized as follows. In Section 2, the formal definition of self-stabilization is presented. The clustering problem is discussed in the Section 3. A robust version of [18] is described in Section 4. The self-stabilization proofs are presented in Section 5. Section 6 discusses about the robustness of our algorithm. Finally, the time complexity is analyzed in Section 7.

2 Model

Communication Model. We model a distributed system by an undirected graph $G=(V,E)$ in which V, is the set of nodes and there is an edge $\{u,v\} \in E$ if and only if the nodes u and v can directly communicate: nodes u and v are said neighbors.

The set of neighbors of a node $v \in V$ will be denoted by N_v. In this paper, we consider the local shared memory model of communication. Each node v has a finite set of *local variables* such that the variables at a node v can be read by node v and the neighbors of v, but can be only modified by node v.

Configuration. The *state* of a node is defined by the values of its local variables. A *configuration* of a distributed system G is an instance of the node states. Let $\mathcal{C}$ be the set of possible configurations.

Program: The program of every node v consists of a finite set of guarded statements of the form *Rule* : *Guard* $\rightarrow$ *Action*.

Guard is a boolean predicate involving the local variables of v and the local variables of its neighbors. *Action* is assignments that modify the local variables in v. If a guard rule is evaluated to true by a node v, then we say the node v *is enabled.*

Computation step. The evaluation of the rule guard, and the action performing is done in an atomic step. The nodes are not synchronized; nevertheless several nodes may perform simultaneously an atomic step. Thus during a

computation step one or several nodes do simultaneously an atomic step.

Computation. A computation e of a system G is a sequence of configurations $c_1, c_2, ...$ such that for $i = 1, 2, ...$, the configuration c_{i+1} is reached from c_i by a single computation step where one or several enabled nodes perform simultaneously an atomic step. $\mathcal{E}$ be the set of all possible computations of a system G. The set of computations of G starting with the particular *initial configuration* $c \in \mathcal{C}$ will be denoted $\mathcal{E}_c$. The set of computations of $\mathcal{E}$ whose initial configurations are all elements of $B \in \mathcal{C}$ is denoted as $\mathcal{E}_B$.

Identifiant. Every node v in the network is assigned a unique identifier (ID). For simplicity, here we identify each node with its ID and we denote both with v.

Attractor. In this paper, we use the notion *attractor* [19] to define self-stabilization.

Definition 1 *(**Attractor**). Let B_1 and B_2 be subsets of $\mathcal{C}$. Then B_1 is an attractor for B_2 if and only if:*

1. $\forall e \in \mathcal{E}_{B_2}, (e = c_1, c_2, ...), \exists i \geq 1 : c_i \in B_1$ *(convergence).*

2. $\forall e \in \mathcal{E}_{B_1}, (e = c_1, c_2, ...), \forall i \geq 1, c_i \in B_1$ *(closure under any computation steps).*

Self-Stabilization. The set of configurations matching the specification of problems is called the set of *legitimate* configurations, denoted as $\mathcal{L}$. $\mathcal{C} \backslash \mathcal{L}$ denotes the set of illegitimate configurations.

Definition 2 *(**Self-Stabilization**). A distributed system S is called Self-Stabilizing if and only if there exists a non-empty set $\mathcal{L} \subseteq \mathcal{C}$ such that the following conditions hold:*

1. $\mathcal{L}$ *is an attractor for* $\mathcal{C}$.

2. $\forall e \in \mathcal{E}_{\mathcal{L}}, e$ *verifies the specification problem.*

T **Stabilization time.** The stabilization time (also named convergence time) is the number of asynchronous rounds needed to reach a legitimate configuration from any initial configuration with any computation.

Definition 3 *(**Asynchronous round**) The asynchronous round of the computation comp $= c_0, ...c_m$ starting at c_i is the smallest segment of comp such that (1) it starts at c_i, and (2) each node enabled at c_i performs a rule during this segment or is not enabled at a configuration of this segment.*

The first asynchronous round of the computation comp is the asynchronous round of comp starting at the initially configuration of comp.

The xth asynchronous round of the computation comp is the asynchronous round starting at the ending configuration of the $x-1$th asynchronous round of comp.

We consider synchronous computation, in which every node performs its code simultaneously.

Definition 4 *A synchronous computation step is a computation step where all enabled nodes perform an action during the step.*
A synchronous computation is a succession of consecutive synchronous computation steps.

Lemma 5 *A single computation step of a synchronous computation is an asynchronous round.*

Proof: Let us study the first computation step of the synchronous computation *comp* starting at c_i: $c_i \xrightarrow{cs} c_{i+1}$. All enabled nodes at c_i perform an action during the computation step cs. Thus c_{i+1} is the ending of the first asynchronous round starting at c_i of *comp* (see the Definition 3). □

2.1 Robustness

The communication graph changes over the time, with node departure, node arrival, communication link failure, network merging, ... G denoted the the communication graph at the current time.

Input changes model. In this paper, we cope with the following types of input changes (these input changes may occur after some failures in the network) : (i) Nodes may quit; for instance, after crash-failure (ii) node may recover or join the network (iii) communication links may fail and/or recovers.

One motivation for our robust stabilization is that a system should react gracefully to the input changes - preserving a safety predicate in the presence of the input changes. The safety predicate is chosen to ensure that the system still performs correctly its task during the period of convergence. A self-stabilizing protocol is robust with respect to input changes, if starting from a safe configuration followed by input changes, the safety predicate holds continuously until the protocol converges to a legitimate configuration.

Definition 6 ***(Robustness under Input Change [19])****. Let $\mathcal{SP}$ be a predicate on configurations called safety predicate, let $\mathcal{IC}$ be a set of input changes in the system. A self-stabilizing distributed system $\mathcal{S}$ is robust under $\mathcal{IC}$ if and only if a set of configurations satisfying the predicate $\mathcal{SP}$ (i) is closed under any computation step, and (ii) is closed under any input change of $\mathcal{IC}$.*

3 Clustering for ad hoc networks

Clustering an ad hoc network means partitioning its nodes into *clusters*, each one with a *clusterhead* and some *ordinary nodes*. In order to meet the requirements imposed by the wireless, mobile nature of these networks, nodes in the same cluster has to be at distance at most 1 of their clusterhead. Thus, the following *clustering property* has to be satisfied:

> **1.** Every ordinary node has at least a clusterhead as neighbor (*dominance* property).

We consider weighted networks, i.e., a weight w_v is assigned to each node $v \in V$ of the network. In ad hoc networks, the amount of bandwidth, memory space or battery power of a node could be used to determine weight values. For simplicity, in this paper we assume that each node has a different weight. Note that if several nodes have the same weight, one may use the couple $(weight, ID)$ to give distinct "weights" to each node. The choice of the clusterheads is based on the *weight* associated to each node: the higher the weight of a node, the better this node is suitable to be a clusterhead.

Assume that the clusterheads are bound to never be neighbors. This implies that, when due to the mobility of the nodes two or more clusterheads become neighbors, those with the smaller weights have to *resign* and affiliate with the now higher neighboring clusterhead. Furthermore, when a clusterhead v becomes the neighbor of an ordinary node u whose current clusterhead has weight smaller than v's weight, u has to affiliate with (i.e., switch to the cluster of) v. These "resignation" and "switching" processes due to node's mobility are a consistent part of the clustering management overhead that should be minimized in ad hoc network where the topology changes fairly often. To overcome the above limitations, in [2] Basagni introduced a generalization of the previous clustering property called *Ad hoc clustering properties* defined as follows:

> **1.** Every ordinary node always affiliates with a neighbor which is clusterhead and has higher weight than the weight of the ordinary node (*affiliation* condition).
>
> **2.** For every ordinary node v, for every clusterhead $z \in N_v : w_z \leq w_{Clusterhead_v} + h$ (*clusterhead* condition).
>
> **3.** A clusterhead has at most k neighboring clusterheads (k being an integer, $0 \leq k < n$) (*k-neighborhood* condition).

The first requirement ensures that each ordinary node has direct access to its clusterhead (the one of the cluster to which it belongs), thus allowing fast inter cluster communication. The second requirement guarantees that each ordinary node always stays with a clusterhead that gives it a "good"

service. By varying the threshold parameter h it is possible to reduce the switching overhead associated to the passage of an ordinary node from its current clusterhead to a new one. When $h = 0$ we simply obtain that each ordinary node affiliates with the neighboring clusterhead with the highest weight. Finally, the third requirement allows us to have up to k clusterheads in its neighboring, $0 \leq k < n$. When $k = 0$ we obtain that two clusterheads can not be neighbors.

3.1 *Safety property for clustering algorithm*

The safety property has to ensure that the network is partitioned into clusters and each cluster has a leader that performs clusterhead tasks. In a clustered network, the role of clusterhead is to act as a local coordinator within a cluster, performing information aggregation and managing communication tasks. Even during the stabilization phase, it is desired that the network is correctly partitioned, i.e., each node belongs to a single cluster having an effectual leader. This property, called "safety", guarantees the functioning of the applications using the hierarchical structure.

Definition 7 *safety property Each node belongs to a single cluster. Each cluster has an effectual leader.*

4 Robust Self-Stabilizing weight based Clustering Algorithm

In this Section, we present a weight-based clustering algorithm : variables are formally presented in Algorithm 1, the predicates and the rules are presented in Algorithm 2. Our algorithm constructs the clusters verifying the ad hoc clustering properties. This algorithm is self-stabilizing and robust to the input changes define in sub-section 4.1. Notice that if $k = h = 0$, our algorithm is a robust and self-stabilizing version of DMAC [3]. Otherwise it is a robust and self-stabilizing version of GDMAC [2].

A node has three possible states. It can be a truly clusterhead, in this case the value of its Ch variable is T. It can be an ordinary node, in this case the value of its Ch variable F. Or, it can be a nearly ordinary node, in this case the value of its Ch variable is NF.

The goal of the R_1 rule is to transform a node v into a well-formed truly clusterhead (i.e., $Ch_v = T \ \wedge \ Clusterhead_v = v$). An ordinary or nearly ordinary node v becomes a clusterhead only when it cannot join a cluster

Algorithm 1 ***: constants and variable definition***

Parameters

k, h : ℕ*;*

Constants

w_v : ℕ*; // the weight of node v*

Local variables of node v

Ch_v*:* {*T, F, NF*}*; // indicates the role of the node v*

$Clusterhead_v$: *IDs; // the clusterhead of node v*

SR_v : ℕ *;//* SR_v *value contains the weight of a v's neighbor.*

Any clusterhead in v's neighborhood having a weight weaker or equal to SR_v *should resign,*

because k-neighborhood condition is violated in v' neighborhood.

neighborhood

Macros

$N_v^+ = \{z \in N_v : (Ch_z = T) \wedge (w_z > w_v)\}$*; // the set of v's neighboring*

clusterheads which have higher weight than v's weight

(i.e., $N_v^+ = \emptyset$). The goal of the R_2 rule is to ensure that the ordinary node v is in a well-formed cluster (i.e., v verifies the *affiliation* and *clusterhead* condition of ad hoc clustering properties). The R_3 action is the first step done by a clusterhead v. After the R_3 action, v is a nearly ordinary node.

A truly clusterhead v ($Ch_v = T$) has to resign its role iff it violates the k-neighborhood condition. A clusterhead v having to resign takes the nearly ordinary state ($Ch_v = NF$) - it performs the R_3 action. Node v stays in this nearly ordinary state until all of nodes in its cluster have joined another cluster. Node v that has a state "nearly ordinary" is requiring that the members of its cluster join another cluster. Thus, the members of v's cluster are enabled (the predicate G_{11} or G_{21} is verified), till v is nearly ordinary. Once the cluster of v is empty (i.e., $\forall z \in N_v : Clusterhead_z \neq v$), node v is enabled; it can become an ordinary node or a truly clusterhead (i.e., the predicate G_1 or G_2 is verified).

A truly clusterhead v checks the number of its neighbors that are clusterheads. If this number is less than or equal to k then SR_v should have the value 0 (R_5 action). If this number is greater than k, then the clusterhead sets up the value of SR_v to the weight of the first neighboring clusterhead having to resign, the

Algorithm 2 ***: Robust Self-Stabilizing Weight-Based Clustering Algorithm***

Predicates

$$\mathbf{G_0}(v) = (\forall z \in N_v^+ : w_v > SR_z) \wedge (|N_v^+| \leq k)$$

$$\mathbf{G_1}(v) = \mathbf{G_{11}}(v) \vee \mathbf{G_{12}}(v)$$

$$\mathbf{G_{11}}(v) \equiv (Ch_v \neq T) \wedge (N_v^+ = \emptyset)$$

$$\mathbf{G_{12}}(v) \equiv (Ch_v = T) \wedge (Clusterhead_v \neq v) \wedge \mathbf{G_0}(v)$$

$$\mathbf{G_2}(v) = \mathbf{G_{21}}(v) \vee \mathbf{G_{22}}(v)$$

$$\mathbf{G_{21}}(v) \equiv (Ch_v = F) \wedge \{(\exists z \in N_v^+ : w_z > w_{Clusterhead_v} + h) \vee (Clusterhead_v \notin N_v^+)\} \wedge (N_v^+ \neq \emptyset)$$

$$\mathbf{G_{22}}(v) \equiv (Ch_v = NF) \wedge \{(\forall z \in N_v : Clusterhead_z \neq v) \wedge (N_v^+ \neq \emptyset)$$

$$\mathbf{G_3}(v) = \mathbf{G_{31}}(v) \vee \mathbf{G_{32}}(v)$$

$$\mathbf{G_{31}}(v) \equiv (Ch_v = T) \wedge \neg\mathbf{G_0}(v)$$

$$\mathbf{G_{32}}(v) \equiv (Ch_v = NF) \wedge (Clusterhead_v \neq v)$$

$$\mathbf{G_4}(v) \equiv (Ch_v \neq T) \wedge (SR_v \neq 0)$$

$$\mathbf{G_5}(v) \equiv (Ch_v = T) \wedge (SR_v \neq max(0,\ k+1^{th}\{w_z : z \in N_v \wedge (Ch_z = T)\}))$$

Rules

$$\mathbf{R_1}(v) : \mathbf{G_1}(v) \rightarrow Ch_v := T;\ Clusterhead_v := v;\ SR_v := max(0,\ k+1^{th}\{w_z : z \in N_v \wedge (Ch_z = T)\})$$

$$\mathbf{R_2}(v) : \mathbf{G_2}(v) \rightarrow Ch_v := F;\ Clusterhead_v := max_{w_z}\{z \in N_v^+\};\ SR_v := 0$$

$$\mathbf{R_3}(v) : \mathbf{G_3}(v) \rightarrow Ch_v := NF;\ Clusterhead_v = v;\ SR_v := 0$$

// update the value of SR_v

$$\mathbf{R_4}(v) : (\neg\mathbf{G_1}(v) \wedge \neg\mathbf{G_2}(v) \wedge \neg\mathbf{G_3}(v)) \wedge \mathbf{G_4}(v) \rightarrow SR_v := 0$$

$$\mathbf{R_5}(v) : (\neg\mathbf{G_1}(v) \wedge \neg\mathbf{G_2}(v) \wedge \neg\mathbf{G_3}(v)) \wedge \mathbf{G_5}(v) \rightarrow SR_v := max(0,\ k+1^{th}\{w_z : z \in N_v \wedge (Ch_z = T)\})$$

one having the (*k+1*)th highest weight (R_5 action). All clusterheads in v's neighborhood having smaller and equal weight than SR_v will have to resign to ensure k-neighborhood condition at node v.

SR_v value of an ordinary node is 0 or v is enabled: the predicate G_1, G_2, G_3

or G_4 is verified.

Due to an incorrect initial configuration, a node v may have to correct the value of its variable $Clusterhead_v$ and/or of its variable SR_v. In this case, v is enabled.

4.1 Safety predicate

The safety predicate $\mathcal{SP}$ is defined as follow:

$$\mathcal{SP} \equiv \forall v \in V : (Clusterhead_v \in N_v \cup \{v\}) \wedge (Ch_{Clusterhead_v} \neq F).$$

$\mathcal{SP}$ predicate ensures that (i) each node belongs to a cluster and that (ii) each cluster has a clusterhead that performs its tasks correctly. Because the nearly ordinary nodes and the truly clusterhead nodes acts as a clusterhead. Thus, the hierarchical structure exists if the predicate $\mathcal{SP}$ is verified.

Let us denote z the clusterhead of a node v. The safety predicate $\mathcal{SP}$ ensures that the node z is a neighbor of node v and node z is not an ordinary node. Thus, the safety predicate $\mathcal{SP}$ is only violated in cases of a z's removal (or a crash of the node z), a failure of link between node v and node z. Therefore, the safety predicate $\mathcal{SP}$ is preserved in the following input changes:

1. Change of node's weight (illustrated in Figure 1).

2. Crash of ordinary nodes.

3. Joining of subnetworks that verify the predicate $\mathcal{SP}$.

4. Failures of link between two ordinary nodes or between two clusterhead nodes.

4.2 Illustration of a convergence phase

Algorithm 2 is illustrated in Figure 1, in this example, $k = 1$ and $h = 0$. Initially, node 5 has 2 clusterheads in its neighborhood. It assigns the value of its SR variable to 9. 9 is the weight of the first clusterhead which violates the 1-neighborhood condition in node 5's neighborhood (Figure 1.b). Node 1 does not stay clusterhead because $SR_5 \geq w_1$: node 1 resigns to nearly ordinary state (Figure 1.c). No node has chosen node 1 as clusterhead (i.e., no node is in the cluster led by node 1). Thus, during the next computation step, node 1 can join the cluster led by node 5. In the neighborhood of node 5 there is one clusterhead, thus node 5 sets the value of its SR variable to 0 (R_5 rule) (Figure 1.d). Due to the change of the weight of node 4 (Figure 1.e), node 2 cannot stay ordinary : all clusterheads in the node 2's neighborhood have a

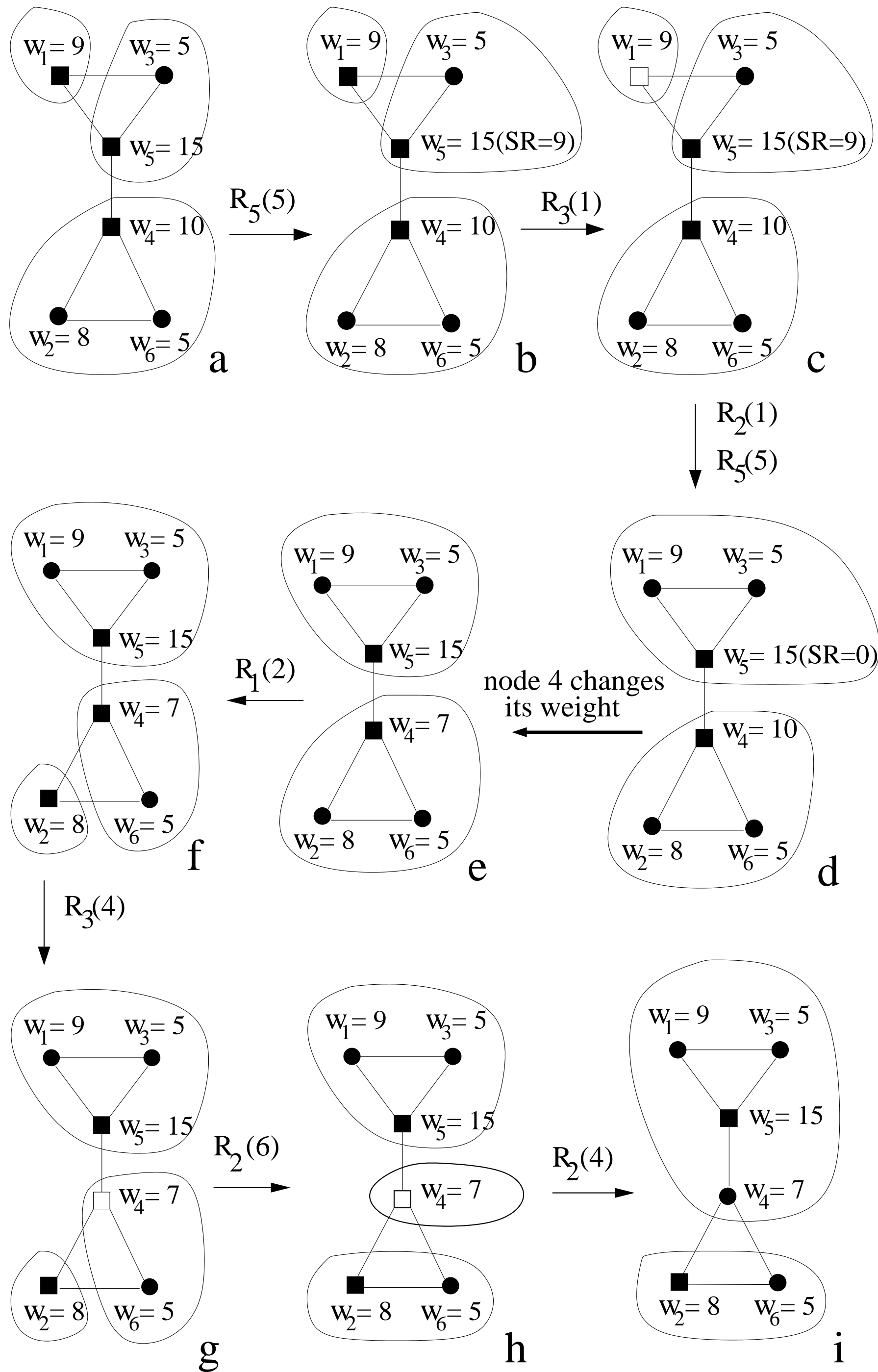

Fig. 1. Convergence to a legitimate configuration in the case k = 1, h = 0

weight that is smaller than node 2's weight. Thus, node 2 becomes clusterhead (Figure 1.f). Node 4 resigns to nearly ordinary state (Figure 1.g). It cannot keep the state "truly clusterhead", because it violates the 1-neighborhood condition: there are two clusterheads in its neighborhood which have a higher

weight than its weight (node 2 and 5). Node 6 does not verify the affiliation condition ($Ch_{clusterhead_6} = Ch_4 = NF$). Node 6 switches of its cluster, it goes into the cluster led by node 2 (Figure 1.h). After that, node 4 can take the state "ordinary" and stop to behave as a clusterhead. Node 4 joins the cluster led by 5 (Figure 1.i). The network is stabilized. During the convergence phase, the safety property $\mathcal{SP}$ is always verified: at any time, the network is partitioned into clusters, and each cluster has a leader ready to do the leadership tasks (i.e., a leader which has the state truly clusterhead or nearly ordinary).

5 Proofs of Self-Stabilization

5.1 Proof of convergence

We first prove that the system reaches a terminal configuration.

Definition 8

- *Let us name P_v the predicate $(Clusterhead_v = v) \vee (Ch_v = F)$.*
- *Let A_1 be the configurations set defined by $\{c \in \mathcal{C} \mid \forall v : P_v \equiv true\}$.*

Lemma 9 *The predicate P_v is closed under any computation step.*

Proof: Assume that we have a computation step $c_1 \stackrel{cs}{\rightarrow} c_2$, such that the predicate P_v is verified by the configuration c_1 and it is not verified by the configuration c_2. Only an action done by node v changes the value of v's variables. Thus node v does an action during the computation step cs. After any action by node v, the predicate P_v is verified. There is a contradiction. □

Lemma 10 *If at the configuration c, the predicate P_v is not verified then node v is enabled at the configuration c.*

Proof: If node v is a truly clusterhead at the configuration c then the predicate $G_{12}(v)$ or the predicate $G_{31}(v)$ is verified by the configuration c. If v is a nearly ordinary node then the predicate $G_{32}(v)$ is verified by the configuration c. In both cases, the node v is enabled in the configuration c. □

Lemma 11 *A_1 is reached after the first asynchronous round.*

Proof: Let *comp* a maximal computation. If the predicate P_v is verified at initial configuration of *comp*, called c_0, then the predicate P_v is verified at the end of the first asynchronous round, because P_v is a closed predicate under any computation step (see Lemma 9). Assume that the predicate P_v is not verified at the configuration c_0. The node v is enabled in the configuration c_0

(see Lemma 10). Two cases are possible :

Case 1: node v does an action during the first round. At the configuration reached after an action by node v, the predicate P_v is verified. The predicate P_v is verified at the end of the first asynchronous round (see Lemma 9).

Case 2: node v does not do an action during the first asynchronous round. At some configuration of the first asynchronous round of *comp*, node v is not enabled (Definition 3 of asynchronous round). We assume without losing any generality, that this configuration is c. The predicate P_v is verified by the configuration c (see Lemma 10). Then, the predicate P_v is verified forever (see Lemma 9).

The predicate P_v is verified at the end of the first round, whatever is node v. $\square$

Corollary 12 *A_1 is an attractor.*

Proof: The configuration set A_1 is closed under any computation step because the predicate P_v is closed under any computation step (see Lemma 9). Along any computation, after an asynchronous round A_1 is reached (see Lemma 11). $\square$

Fact 13 *In any configuration of A_1, the predicate $G_{12}(v)$ and the predicate $G_{32}(v)$ are never true.*

Lemma 14 *In A_1, along any maximal computation, between two consecutive R_1 actions by node v, another node u such that $w_u > w_v$, does the R_1 action.*

Proof: Let us study the segment of a computation $c_1, c_2, \ldots, c_m$ starting and ending by a computation step where v does the R_1 action. At the configuration c_1 and the configuration c_{m-1} the predicate $G_{11}(v)$ is verified, we have $N_v^+ = \emptyset$.
Once node v had performed the R_1 action, P_v is verified and v is a truly clusterhead. Before performing again the R_1 action, v becomes a nearly ordinary node. Thus, node v does the R_3 action during the segment $c_2, \ldots, c_{m-1}$.
Assume that the R_3 action of v is done during the computation step $c_i \xrightarrow{cs} c_{i+1}$ where $1 < i < m$. The predicate $G_{31}(v)$ is verified by c_i, thus we have $N_v^+ \neq \emptyset$ at c_i. Thus between c_1 and c_i, a node $u \in N_v$, such that $w_u > w_v$ has performed the R_1 action. $\square$

Lemma 15 *Along any maximal computation starting from a configuration of A_1, a node v performs a finite number of times the R_1 action.*

Proof: Assume that *comp* is a maximal computation where the node v executes infinitely often the R_1 action. Following Lemma 14, between two consecutive the R_1 action by node v a node u such that $w_u > w_v$ performs the R_1 action. Since the set of nodes is finite, then v performs the R_1 action, infinitely often only if there exists a node u ($w_u > w_v$) that performs the R_1 infinite often. Thus, we have an infinite sequence of nodes having increasing weight that perform R_1 action infinitely often. Since the number of nodes is finite, this is a contrary. Hence our hypothesis is false. From A_1, along any maximal computation a node executes a finite number of times the R_1 action. □

Lemma 16 *In A_1, along any maximal computation, between two consecutive R_3 actions by node v, v does the R_1 action.*

Proof: Once node v had performed the R_3 action, v is a nearly ordinary node. Before performing R_3 action, the predicate $G_{31}(v)$ has to be verified. Thus, node v needs to become a truly clusterhead in the meantime. Only the R_1 action transforms an (nearly) ordinary node into a truly clusterhead. Thus between two consecutive R_3 actions by node v, v does the R_1 action. □

Lemma 17 *In A_1, along any maximal computation a node v performs a finite number of times the R_2 action.*

Proof: Assume that *comp* is a maximal computation where the node v executes infinitely often R_2 action. *comp* has a suffix where node v does not execute the R_1 action and the R_3 action but executes infinitely often the R_2 action. Let us study the action of v in this suffix. Once v had performed the R_2 action, node v is an ordinary node. We have $(\forall z \in N_v^+ : w_z \leq w_{Clusterhead_v}) \wedge (Clusterhead_v \in N_v^+)$.

When the node v performs R_2 action, the predicate $G_{21}(v)$ is verified. We have $(\exists z \in N_v^+ : w_z > w_{Clusterhead_v} + h) \vee (Clusterhead_v \notin N_v^+)$, implies that in meantime $Clusterhead_v$ has performed the R_3 action or a neighbor of v, z has became a truly clusterhead. Thar is a contrary. □

Corollary 18 *Every maximal computation comp that starts in A_1 has a suffix where only the R_4 action and the R_5 action are executed.*

Proof: During maximal computation *comp*, the number of R_1 actions, R_3 actions and R_2 actions are finite (see Lemmata 15, 16, and 17). □

Theorem 19 *Starting from a configuration of A_1, any maximal computation reaches a terminal configuration.*

Proof: Let us study a maximal computation *comp. comp* has a suffix where only the R_4 actions and the R_5 actions are executed (see Corollary 18). In this suffix, named suf, each node does at most one time the R_4 action or R_5

action. Because once the predicate $G_5 \vee G_4$ is not verified by a node, it will be never verified along suf. Thus, suf contains at most $|V|$ computation steps. We conclude that $comp$ reaches a terminal configuration. □

5.2 Proof of correctness

In this Section we prove that all terminal configurations are legitimate.

Lemma 20 *Let c be a configuration that contains a nearly ordinary node. c is not a terminal configuration.*

Proof: Assume that node v is a nearly ordinary node (i.e., $Ch_v = NF$ is verified). If $\forall u \in N_v,\ Clusterhead_u \neq v$ is verified then the predicate $G_{11}(v)$ or the predicate $G_{22}(v)$ is verified. In this case, node v is enabled at the configuration c. Assume that there is a node $u \in N_v$ such that $Clusterhead_u = v$.

Case 1: node u is ordinary. Since $Ch_v = NF$ then $Clusterhead_u \notin N_u^+$ (see the definition of N_u^+). Thus, the predicate $G_{21}(u)$ is verified. Node u is enabled at the configuration c.

Case 2: node u is a truly clusterhead. We have $Ch_u = T$. Since $Clusterhead_u = v \neq u$. Thus, the predicate $G_{12}(u)$ or the predicate $G_{31}(u)$ is verified. Node u is enabled at the configuration c.

Case 3: node u is nearly ordinary. We have $Ch_u = NF$. Since $Clusterhead_u = v \neq u$. Thus, the predicate $G_{32}(u)$ is verified. Node u is enabled at the configuration c. □

Theorem 21 *In a terminal configuration, the ad hoc clustering properties are satisfied.*

Proof: In a terminal configuration, for every node v, we have $G_i(v) \equiv False :\ i = \{1..5\}$. Following Lemma 20, in a terminal configuration there is not a node v such that $Ch_v = NF$.

Case 1: node v is ordinary, we have $Ch_v = F$. The predicate $G_1(v)$ is not verified implies that N_v^+ is not empty. The predicate $G_2(v)$ is not verified implies that $(\nexists z \in N_v^+ : (w_z > w_{Clusterhead_v} + h))$ and $(Clusterhead_v \in N_v^+)$. Thus node v satisfies affiliation and clusterhead condition (properties 1 and 2).

Case 2: node v is a truly clusterhead, we have $Ch_v = T$. The predicate $(G_3(v)$ is not verified implies that $(\forall z \in N_v^+ : w_v > SR_z) \wedge (|N_v^+| \leq k)$.

The predicate $G_1(v)$ is not verified implies that $Clusterhead_v = v$. We now prove that node v has at most k neighboring clusterheads. Since $|N_v^+| \leq k$, then node v has at most k neighboring clusterheads with higher weight than v's weight. Assume that node v has more than k neighboring clusterheads. The $k+1$th of these clusterheads has a weight smaller than v weight. If $SR_v \neq k+1^{\text{th}}\{w_z : z \in N_v \wedge (Ch_z = T)\})$ then node v is enabled. Thus, $SR_v = k+1^{\text{th}}\{w_z : z \in N_v \wedge (Ch_z = T)\}) < w_v$ and v has a neighboring clusterhead u such that $w_u \leq SR_v$. Hence, the predicate $G_{31}(u)$ is verified because $v \in N_u^+$ and $w_u \leq SR_v$. Node u is enabled. That is a contrary. □

6 Robustness

In a configuration that satisfies the predicate $\mathcal{SP}$, the clusterhead of any node performs its task correctly, because it is not an ordinary node. Thus, the hierarchical structure is kept up. Let us remind the definition of the predicate $\mathcal{SP}$: $\mathcal{SP} \equiv \forall v \in V : (Clusterhead_v \in N_v \cup \{v\}) \wedge (Ch_{Clusterhead_v} \neq F)$.

Definition 22 *Let v a node. We define $\mathcal{SP}_v$ as the safety predicate $\mathcal{SP}$ on v. $\mathcal{SP}_v \equiv (Clusterhead_v \in N_v \cup \{v\}) \wedge (Ch_{Clusterhead_v} \neq F)$.*

Lemma 23 *The predicate $\mathcal{SP}_v$ is closed any computation step.*

Proof: Assume that we have a computation step $c_1 \xrightarrow{cs} c_2$, we will prove that if the predicate $\mathcal{SP}_v$ is verified by the configuration c_1, then in the configuration c_2, the predicate $\mathcal{SP}_v$ is verified.
We will prove by contrary. Assume that in the configuration c_2, the predicate $\mathcal{SP}_v$ is not verified: $(Clusterhead_v \notin \{N_v \cup v\}) \vee (Ch_{Clusterhead_v} = F)$. Thus, during the computation step cs, there are two possibilities.

Case 1. node v changes its clusterhead during the execution of cs. Note that the R_4 actions and R_5 actions do not change the value of clusterhead of node v. If node v performs the R_1 action or R_3 action during the computation step cs then the predicate $\mathcal{SP}_v$ is always verified because after doing the action of the rule R_1 or R_3, we have $(Clusterhead_v = v) \wedge (Ch_v \neq F)$. Thus, node v performs the R_2 action during the computation step cs. We denote z the clusterhead selected by node v during the computation step cs. In the configuration c_1, we have $Ch_z = T$ and in the configuration c_2, we have $Ch_z = F$. During the computation step cs, the node z cannot perform the R_2 action. Thus, there is a contrary because the rule R_2 is the only rule that changes the value of the variable Ch to F.

Case 2. node v did not change its clusterhead during the computation step cs. We denote z the clusterhead of node v. In the configuration c_1, the predicate

$\mathcal{SP}_v$ is verified implies that $Ch_z \neq F$. In the configuration c_2, the predicate $\mathcal{SP}_v$ is not verified implies that $Ch_z = F$. Thus, during the execution of cs, the node z performed R_2 action. But the node z can perform R_2 action only when the predicate $G_{22}(z)$ is verified, that implies $Clusterhead_v \neq z$ in the configuration c_1. That is a contrary. □

Theorem 24 *The predicate $\mathcal{SP}$ is closed under any computation step.*

Proof: The theorem follows directly from Lemma 23. □

7 Time complexity

7.1 Time to reach a safe configuration

In this Section, we study the time that is needed to reach a safe configuration. A safe configuration verifies the predicate $\mathcal{SP}$. We prove that along any computation, a safe configuration is reached in a single synchronous computation step.

Lemma 25 *Assume that in the configuration c, we have the predicate $G_i(v) \equiv False$, $\forall i \in \{1..3\}$. The predicate $\mathcal{SP}_v$ is verified by the configuration c.*

Proof:

1. v **is a truly clusterhead**. Since the predicates $G_{12}(v)$ and $G_{31}(v)$ are not verified, we have $Clusterhead_v = v$, thus the predicate $\mathcal{SP}_v$ is verified by the configuration c.

2. v **is an ordinary node**. Since the predicates $G_{11}(v)$ and $G_{21}(v)$ are not verified, we have $Clusterhead_v \in N_v^+$, thus the predicate $\mathcal{SP}_v$ is verified by the configuration c.

3. v **is an nearly ordinary node**. Since the predicate $G_{32}(v)$ is not verified, we have $Clusterhead_v = v$, thus the predicate $\mathcal{SP}_v$ is verified by the configuration c.

Thus, in any case, the predicate $\mathcal{SP}_v$ is verified by the configuration c. □

Lemma 26 *If during a computation step, node v does an action. The predicate $\mathcal{SP}_v$ is verified by the configuration reached after the computation step.*

Proof: Let us study the computation step $c_1 \stackrel{cs}{\rightarrow} c_2$ where node v does an action.

1. node v performs the R_1 action during the computation step cs. After performing $R_1(v)$ action, we have $(Clusterhead_v = v) \wedge (Ch_v = T)$, thus the predicate $\mathcal{SP}_v$ is verified by the configuration c_2.

2. node v performs R_2 action during the computation step cs. We denote z' the clusterhead selected by node v during the computation step cs. In the configuration c_1, z' is a truly clusterhead. Assume that the configuration c_2 does not verify the predicate $\mathcal{SP}_v$ (i.e. in the configuration c_2, z' is an ordinary node). During the execution of cs, the node z' has performed R_2 action. But the node z' can perform R_2 action only if the predicate $G_2(z')$ is verified by the configuration c_1, that implies that $Ch_{z'} \neq T$ in the configuration c_1. That is a contrary.

3. node v performs R_3 action during the computation step cs. $G_3(v) \equiv True$ in c_1. After performing $R_3(v)$ action, we have $(Clusterhead_v = v) \wedge (Ch_v = NF)$, thus, the predicate $\mathcal{SP}_v$ is verified by the configuration c_2.

4. node v performs the R_4 action or the R_5 action during cs. In the configuration c_1, $G_i(v) \equiv False$, $\forall i \in \{1..3\}$. The predicate $\mathcal{SP}_v$ is verified by the configuration c_1 (see Lemma 25). Since the predicate $\mathcal{SP}_v$ is closed under any computation step (Lemma 23), then in the configuration c_2, the predicate $\mathcal{SP}_v$ is verified.

Thus, in any case, the predicate $\mathcal{SP}_v$ is verified by the configuration c_2. $\square$

Theorem 27 *The system verifies the predicate $\mathcal{SP}$ after the first asynchronous round of any computation.*

Proof: Let us study the computation $c_0, c_1, ...c_i$. Without losing any generality, we assume that the first asynchronous round is $comp' = c_0, \ldots, c_m$. We prove that in the configuration c_m, the predicate $\mathcal{SP}_v$ is verified, for every node v.

Case 1. In the configuration c_0, $G_i(v) \equiv False$, $\forall i \in \{1..3\}$. The predicate $\mathcal{SP}_v$ is verified by the configuration c_0 (see Lemma 25). Since the predicate $\mathcal{SP}_v$ is closed under any comptation step (Lemma 23), then the predicate $\mathcal{SP}_v$ is verified by the configuration c_m.

Case 2. In the configuration c_0, $\exists i \in \{1..3\}$: $G_i(v) \equiv True$.

Case 2.1 During a computation step of $comp'$, node v performs an action. We assume without losing any generality, that this action is done during the computation step $c_i \xrightarrow{cs} c_{i+1}$ where $i < m$. The predicate $\mathcal{SP}_v$ is verified by the configuration c_{i+1} (see Lemma 26), thus it is verified by the configuration c_m ($\mathcal{SP}_v$ is a closed predicate under any computation step).

Case 2.2 During any computation step of $comp'$, node v does not do an action. At some configuration of $comp'$, node v is not enabled (by definition of the first asynchronous round). We assume without losing any generality, that

this configuration is c_i where $0 \leq i \leq m$. The predicate $\mathcal{SP}_v$ is verified by the configuration c_i (see Lemma 25), then it is verified by the configuration c_m.

We conclude that the predicate $\mathcal{SP}$ is verified by the configuration c_m. □

Corollary 28 *The system verifies the predicate $\mathcal{SP}$ after the first computation step of a synchronous computation.*

Proof: During a synchronous computation, a single computation step is an asynchronous round (see Lemma 5). According to Theorem 27, after an asynchronous round, a safe configuration is reached. □

7.2 Convergence time

The stabilization time (or convergence time) is the maximum number of asynchronous rounds needed to reach a legitimate configuration from an arbitrary initial one. We will establish that along any computation, a legitimate configuration is reached in less than $2|V| + 3$ asynchronous rounds.

To compute the stabilization time we need to define V_i, a set of nodes for $0 < i \leq |V|$, as follows:

Definition 29 *DAG =(V', E') is the Directed Acyclic Graph built on G=(V,E) as follows:*

- *$V' = V$ is the set of nodes in the initial distributed system.*
- *E' is the arrows set. The arrow $v \to u$ belongs to E' if and only if $w_v > w_u$ and $(u, v) \in E$.*

Definition 30

- *Set_1 is the set of DAG sources. A source is a node with no incoming edges in the DAG.*
- *$V_0 = \emptyset$.*
  for $i > 0$, $V_i = \bigcup_{j=1}^{j=i} Set_j$.
- *All the parents of a node, v_{i+1}, of Set_{i+1} belong to V_i and the node v_{i+1}, does not belong to V_i. Formally, for $i \geq 1$, $S_{i+1} = \{ v \notin V_i \mid (u \to v) \Rightarrow u \in V_i\}$.*

Remark 31 *Let us name l the length of the DAG. l is the length of the largest directed path.*
We have $l \leq D$ where D is the network diameter; and $V_{l+1} = V$.

We will establish that from a configuration of A_1, after $2l + 2$ asynchronous rounds, no node performs the R_1 action or R_3 action.

Lemma 32 *From a configuration of A_1, a node of V_0 will never perform the R_1 action or R_3 action, along any computation. The value SR of a truly clusterhead of V_0 can only decrease from a configuration of A_1.*

Proof : No node of V_0 perform the R_1 action or R_3 action. The value SR of a truly clusterhead of V_0 can only decrease. The both facts are true because V_0 is empty. □

Lemma 33 *Let i be an integer greater than 0. From a configuration of A_1, after $2i-1$ asynchronous rounds, a node of V_i will never perform the R_1 action or R_3 action, along any computation. The value SR of a truly clusterhead of V_i can only decrease after $2i$ asynchronous rounds from a configuration of A_1.*

Proof : The proof is done by induction. By hypotheses, we have (1) no node of V_{i-1} performs the R_1 action or R_3 action after sup($2i-3$,0) asynchronous rounds from a configuration of A_1 and (2) the value SR of a truly clusterhead of V_{i-1} can only decrease after $2i-2$ asynchronous rounds from a configuration of A_1.

Let v_i be a node of Set_i. After $2i-2$ asynchronous rounds from a configuration of A_1,

- The nodes of $N_{v_i}^+$ are neighbor of v_i and their weight are higher than the weight of v_i (by definition of $N_{v_i}^+$). Thus, the nodes of $N_{v_i}^+$ belong to V_{i-1}. The set $N_{v_i}^+$ is stable (i.e. it will never change).
- The value SR of a truly clusterhead of V_{i-1} can only decrease (by induction hypothesis). Thus, if $G_0(v_i)$ is verified then it will be always verified along any computation.
- If $N_{v_i}^+$ is empty then $G_0(v_i)$ is verified.
- If $G_0(v_i)$ is not verified then $N_{v_i}^+$ is not empty, and it will never become empty.
- If $G_{11}(v_i)$ is not verified, it will be not verified along any computation. Because, (1) $N_{v_i}^+$ is not empty it will never be empty; or (2) the node v_i is a truly clusterhead that will never give up its status (because $G_0(v_i)$ is always verified).
- if $G_{31}(v_i)$ is not verified, it will be not verified along any computation. Because, (1) $G_0(v_i)$ is verified, it will be always verified along any computation; or (2) the node v_i is not and will never become a truly clusterhead ($N_{v_i}^+$ is never empty).
- If the predicate $G_{11}(v_i)$ is verified, then $G_0(v_i)$ is verified and v_i is enabled. Therefore, the node v_i performs the R_1 action during the $2i-$ 1th asynchronous round. At the end of this round, $G_{11}(v_i)$ and $G_{31}(v_i)$ are not verified.

- If the predicate $G_{31}(v_i)$ is verified, then $N^+_{v_i}$ is not empty and v_i is enabled. Therefore, the node v_i performs the R_3 action during the $2i-1$th asynchronous round. At the end of this round, $G_{11}(v_i)$ and $G_{31}(v_i)$ are not verified.

The predicates $G_{12}(v_i)$ and $G_{32}(v_i)$ are not verified by the node v_i in a configuration of A_1. Thus, after $2i-1$ asynchronous rounds from a configuration of A_1, along any computation, v_i will not perform the R_1 action or R_3 action. Moreover, no neighbor of a truly clusterhead of V_i will become a clusterhead.

Assume the value SR of a truly clusterhead of V_i, named u_i, increases during the xth asynchronous round ($x > 2i$) (i.e. the node u_i has performed the R_5 action during the xth asynchronous round). A neighbor of u_i has becomed a truly clusterhead during or after the $x-1$th asynchronous round. Thus ($x-1 \leq 2i$). There is a contradiction. We conclude that after $2i$ asynchronous rounds from a configuration of A_1, along any computation, the value SR of a truly clusterhead of V_i can only decrease. □

We have proved that a configuration of A_1 is reached after a single asynchronous round (see Lemma 11) from any configuration along any computation.

Corollary 34 *After $2l+2$ asynchronous rounds from any configuration along any computation, no node will perform the R_1 action or R_3 action.*

Theorem 35 *After $2l+3$ asynchronous rounds from any configuration along any computation, the predicates $G_4(v)$, G_5, and $G_{21}(v)$ are never verified.*

Proof : After $2l+2$ asynchronous rounds from any configuration, if the predicate $G_4(v)$ (resp. predicate $G_5(v)$, predicate $G_{21}(v)$) is not verified then it will never be verified because N^+_v is stable, no node become nearly ordinary, and only v may change the value of SR_v.

After $2l+2$ asynchronous rounds from any configuration, if the predicate $G_4(v)$ (resp. predicate $G_5(v)$, predicate $G_{21}(v)$) is true, then the node v is enabled. Therefore v performs the R_4 action (resp. R_5 action, R_{21} action) is done during the $2l+3$th round. At the end of this asynchronous round, the predicates $G_4(v)$, G_5, and $G_{21}(v)$ are verified. □

Theorem 36 *After $2l+4$ asynchronous rounds from any configuration along any computation, no node will perform an action.*

Proof : After $2l+3$ asynchronous rounds from any configuration along any computation, only the nearly ordinary nodes are enabled (see Corollary 34 and Lemma 35). No node verifies the predicate G_{21}. Thus a nearly ordinary node, v, is the leader of an empty cluster: v is enabled (i.e. $G_{11}(v)$ or $G_{22}(v)$

is verified). v cannot verify the $G_{11}(v)$ predicate (see proof of Lemma 33).

Thus, the node v verifies the predicate $G_{22}(v)$; v does the R_2 action during the $2l + 4$th round. At the end of this asynchronous round, v is ordinary, and it will stay ordinary (because, it never perform the R_1 action). □

We conclude that a terminal configuration is reached after at most $2|D| + 4$ asynchronous rounds along any computation, from any initial configuration.

Figure 2 illustrates the number of asynchronous rounds needed to stabilize in the case $k = 1,\ h = 0$.

Note that this example can be generalized at any value of k and h. We have a configuration c composed of m blocs as depicted in Figure 2. Each bloc B_i includes 3 nodes: X_i, Y_i, Z_i. We assume that the node weights are ordered as the following: $X_i > Y_i > Z_i > Y_{i+1}$. We denote $|V|$ the number of nodes in the system S, $|V| = 3m$. Notice that the diameter of the system is equal to $2m$. Following Algorithm 2, from the initial configuration, each bloc B_i will one after another takes three asynchronous rounds to stabilize. Thus, $3m = 3D/2$ asynchronous rounds are needed to converge to a legitimate configuration. Notice that if $k \geq \Delta$, where Δ is the maximal degree of the network, the k-neighborhood condition is always verified. Thus, the convergence time is $O(1)$ rounds under an asynchronous scheduler.

8 Conclusion

In this paper, we present a robust and self-stabilizing version of GDMAC and DMAC. Starting from an arbitrary configuration, the system satisfies the safety predicate in one synchronous computation step (i.e., one asynchronous round). Once the system satisfies the safety predicate, the system performs correctly its task (i.e., the network is partitioned into clusters). During the construction of the final clusters the safety predicate stay verified : the network is always partitioned. Once a terminal configuration is reached, the *ad hoc clustering properties* are satisfied. Moreover, our algorithm could be applied to several 1-hop clusters formation solutions in [4,9,12,15].

We have established that the stabilization time is at most $O(D)$ asynchronous rounds, where D is the network diameter.

Our algorithm is designed for the state model. Nevertheless, it can be easily transformed into an algorithm for the message-passing model. For this purpose, each node v periodically broadcasts to its neighbors a message containing its state. Based on this message, v's neighbors decide whether to update their

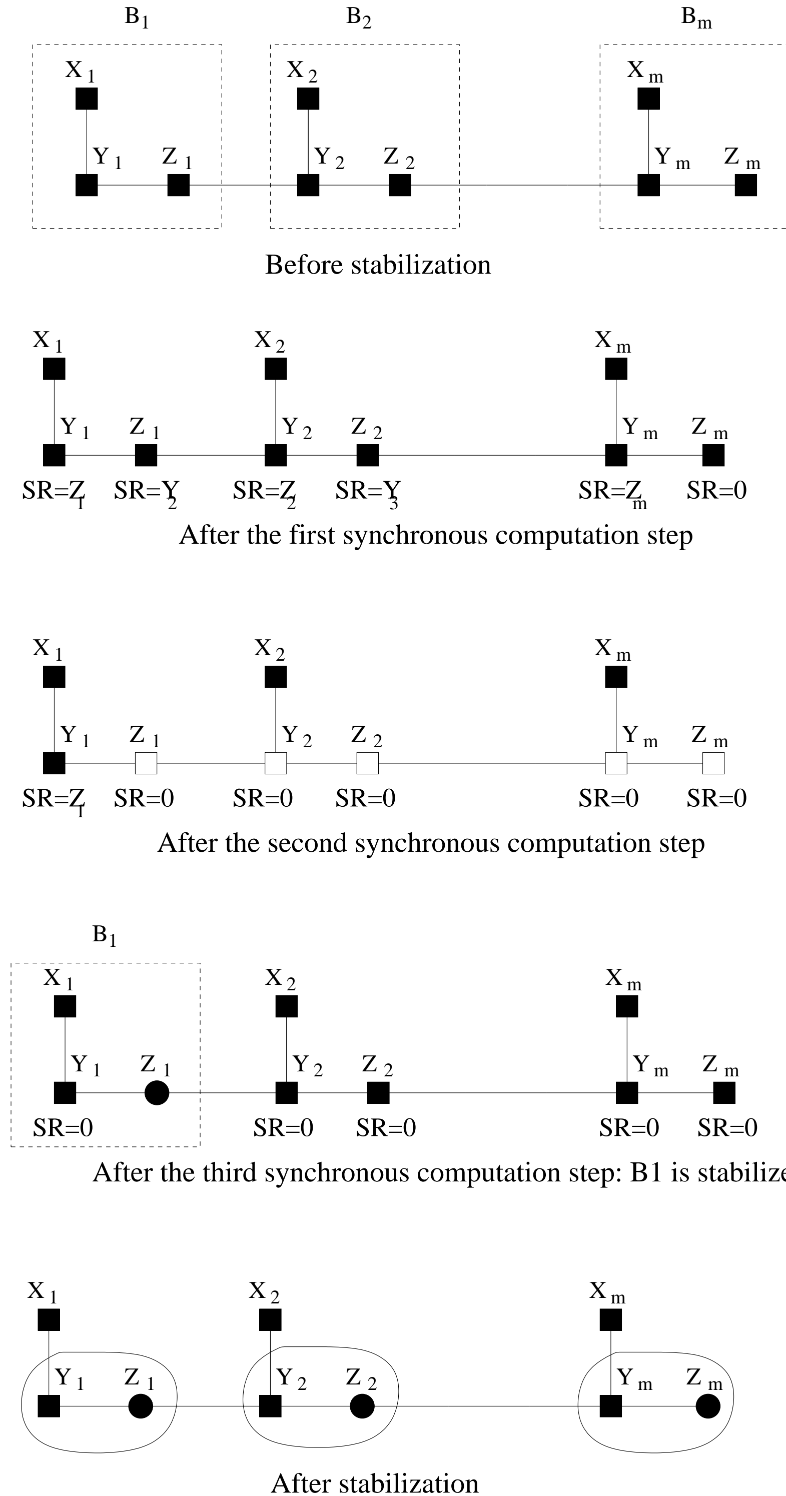

Fig. 2. Stabilization time in the case k = 1, h = 0

variables or not. After a change in the value of v's state, node v broadcasts to its neighbors its new state.

References

[1] S. Banerjee and S. Khuller. A clustering scheme for hierarchical control in multi-hop wireless networks. In *the 20th IEEE Conference on Computer Communications (INFOCOM'01)*, pages 1028–1037, 2001.

[2] S. Basagni. Distributed and mobility-adaptive clustering for multimedia support in multi-hop wireless networks. In *the IEEE* 50*th International Vehicular Technology Conference (VTC'99)*, pages 889–893, 1999.

[3] S. Basagni. Distributed clustering for ad hoc networks. In *the International Symposium on Parallel Architectures, Algorithms, and Networks (ISPAN'99)*, pages 310–315, 1999.

[4] P. Basu, N. Khan, and T. Little. A mobility based metric for clustering in mobile ad hoc networks. In *the 21st International Conference on Distributed Computing Systems (ICDCSW '01)*, page 413, 2001.

[5] D. Bein, A. K. Datta, C. R. Jagganagari, and V. Villain. A self-stabilizing link-cluster algorithm in mobile ad hoc networks. In *the 8th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'05)*, pages 436–441, 2005.

[6] C. Bettstetter and B. Friedrich. Time and message complexities of the generalized distributed mobility-adaptive clustering (GDMAC) algorithm in wireless multihop networks. In *the IEEE Vehicular Technology Conference (VTC'03)*, pages 176–180, 2003.

[7] C. Bettstetter and R. Krausser. Scenario-based stability analysis of the distributed mobility-adaptive clustering (DMAC) algorithm. In *the 2nd ACM Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc'01)*, pages 232–241, 2001.

[8] M. Chatterjee, S. Das, and D. Turgut. WCA: A weighted clustering algorithm for mobile ad hoc networks. *Journal of Cluster Computing, Special issue on Mobile Ad hoc Networking*, 5(2):193–204, 2002.

[9] C. Chiang, H. Wu, W. Liu, and M. Gerla. Routing in clustered multihop, mobile wireless networks with fading channel. In *the IEEE SICON'97*, pages 197–211, 1997.

[10] S. Dolev and N. Tzachar. Empire of colonies self-stabilizing and self-organizing distributed algorithms. In *the 10th International Conference On Principles Of Distributed Systems (OPODIS'06), Springer LNCS 4305*, pages 230–243, 2006.

[11] V. Drabkin, R. Friedman, and M. Gradinariu. Self-stabilizing wireless connected overlays. In *the 10th International Conference On Principles Of Distributed Systems (OPODIS'06), Springer LNCS 4305*, pages 425–439, 2006.

[12] A. Ephremides, J.E. Wieselthier, and D.J. Baker. A design concept for reliable mobile radio networks with frequency-hopping signaling. *In the IEEE Transactions on Wireless communications*, pages 56–73, 1987.

[13] Y. Fernandess and D. Malkhi. K-clustering in wireless ad hoc networks. In *the 2nd ACM international workshop on Principles of mobile computing (POMC'02)*, pages 31–37, 2002.

[14] M. Frodigh, P. Johansson, and P. Larsson. Wireless ad hoc networking: The art of networking without a network. In *Ericsson Review, No. 4*, 2000.

[15] M. Gerla and J. T. Tsai. Multicluster, mobile, multimedia radio network. *Wireless Networks*, 1(3):255–265, 1995.

[16] W. Goddard, S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Self-stabilizing protocols for maximal matching and maximal independent sets for ad hoc networks. In *the 5th IPDPS Workshop on Advances in Parallel and Distributed Computational Models (WAPDCM'03)*, 2003.

[17] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *In the IEEE Transactions on Wireless communications*, 1(4):660–670, 2002.

[18] C. Johnen and L. Nguyen. Self-stabilizing weight-based clustering algorithm for ad hoc sensor networks. In *the 2nd International Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors'06), Springer LNCS 4240*, pages 83–94, 2006.

[19] C. Johnen and S. Tixeuil. Route preserving stabilization. In *the 6th International Symposium on Self-stabilizing System (SSS'03), Springer LNCS 2704*, pages 184–198, 2003.

[20] H. Kakugawa and T. Masuzawa. A self-stabilizing minimal dominating set algorithm with safe convergence. In *the 8th IPDPS Workshop on Advances in Parallel and Distributed Computational Models (APDCM'06)*, 2006.

[21] C. R. Lin and M. Gerla. Adaptive clustering for mobile wireless networks. *In the IEEE Journal on Selected Areas in Communications*, 15(7):1265–1275, 1997.

[22] N. Mitton, A. Busson, and E. Fleury. Self-organization in large scale ad hoc networks. In *the 3rd Annual Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET'04)*, June 2004.

[23] N. Mitton, E. Fleury, I. Guérin. Lassous, and S. Tixeuil. Self-stabilization in self-organized multihop wireless networks. In *the 25th IEEE International Conference on Distributed Computing Systems Workshops (WWAN'05)*, pages 909–915, 2005.

[24] Z. Xu, S. T. Hedetniemi, W. Goddard, and P. K. Srimani. A synchronous self-stabilizing minimal domination protocol in an arbitrary network graph. In *the 5th International Workshop on Distributed Computing (IWDC'03), Springer LNCS 2918*, 2003.

[25] O. Younis and S. Fahmy. Distributed clustering for ad-hoc sensor networks: A hybrid, energy-efficient approach. In *the 23rd IEEE Conference on Computer Communications (INFOCOM'04)*, 2004.