Self-Stabilizing Bounded Size Clustering Algorithm

Rapport de Recherche LRI n^o 1464

Colette Johnen, Le Huy Nguyen
Univ. Paris-Sud, CNRS
LRI, Univ. Paris-Sud, F-91405, Orsay Cedex, France
E-mail: colette@lri.fr, lehuy@lri.fr

Abstract

Clustering means partitioning nodes into groups called clusters, providing the network with a hierarchical organization. Overall, clustering increases the scalability of network management. For instance, clustering-based routing reduces the amount of routing information propagated in the network; members of a cluster can share resources; and clustering can be used to reduce the amount of information to store the network state.

A self-stabilizing algorithm, regardless of the initial system state, automatically converges to a set of states that satisfy the problem specification without external intervention. Due to this property, self-stabilizing algorithms are adapted to highly dynamic networks as ad hoc or sensors networks. From a configuration resulting of topological changes, the system will automatically converge to a configuration consistent with the new topology.

In this paper, we present the first Self-stabilizing Clustering Algorithm where the obtained clusters have a bounded size.

Keywords: Self-stabilization, Distributed algorithm, Clustering, Ad hoc network, sensors network, bounded size cluster.

Résumé

Le problème d'agrégation consiste à partitionner les noeuds d'un réseau en grappes, donc de donner au réseau une organisation hiérarchique. L'agrégation facilite la gestion de réseau de très grande taille. Par exemple, les tables de routages sont plus petites dans le cas d'un protocole de routage basé sur les grappes/clusters, ainsi que la quantité de l'information échangée en vu du routage.

Un algorithme auto-stabilisant, indépendant de l'état initial du système, converge vers des états où le système fonctionn correctement dans un temps fini. Grâce à cette propriété, les algorithmes auto-stabilisants sont bien adapté aux réseaux dont la topologie fluctue tel que les réseaux ad-hoc et les réseaux de sensors. Après un changement de topologie, le système converge automatiquement vers un fonctionnement adapté à la nouvelle topologie.

Dans cet article, nous présentons un algorithme auto-stabilisant d'agrégation où les grappes ont une taille bornée.

Mots-clés: Auto-stabilization, Algorithme distribué, Agrégation, Réseau ad-hoc, réseau de sensors, grappes de taille bornée.

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 fashion, or may be connected to the larger Internet [7]. 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 create 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 networks composed of a clusterhead and ordinary nodes. Each node belongs to only one cluster. In addition, a cluster is required to obey to 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. 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 detailed state 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, 2, 3, 5, 6, 8, 13]. The clustering algorithms in [1, 6] 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 [8, 13] 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 [5], 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 selected nodes are those that are the most suitable for the clusterhead role (i.e., a node optimizing all the parameters). 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 is 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].

A system is self-stabilizing when regardless of its initial configuration, it is guaranteed to reach a legitimate configuration in a finite number of steps. A system which is not self-stabilizing may stay in an illegitimate configuration forever. 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. Self-stabilizing algorithms are thus, inherently tolerant to transient faults in the system. Many 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 changes is viewed as an inconsistent configuration from which the system will con-

verge to a configuration consistent with the new topology. Several self-stabilizing algorithms for cluster formation and clusterhead selection have been proposed [4, 9, 10, 12, 14, 15]. [12] presents a robust self-stabilizing version of DMAC under the synchronous schedule. A self-stabilizing version of DMAC and GDMAC is presented in [10]. A robust and self-stabilizing version of GDMAC is presented in [9]. In [4], a self-stabilizing link-cluster algorithm under an asynchronous message-passing system model is 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 algorithm adapted to message passing model requires one type of messages. A self-stabilizing algorithm for cluster formation is presented in [15]. A density criteria (defined in [14]) is used to select clusterhead: a node v chooses in its neighborhood the node having the highest density. A v's neighborhood contains all nodes at distance less or equal to 2 from v. Therefore, to choose clusterhead, communication at distance 2 is required. Our algorithms build clusters on local information; thus, it requires only communication between nodes at distance 1 of each others.

Contribution: The approaches for finding the clusterheads do not produce an optimal solution with respect to power usage and load balancing. In this paper, we propose a self-stabilizing clustering algorithm which takes into consideration the number of nodes a clusterhead can handle, and transmission power, mobility, and battery power of the nodes. Our algorithm guarantees that network nodes are partitioned into clusters: each cluster has at most $SizeBound$ nodes. Thus, our algorithm achieves load balancing by guarantee a threshold ($SizeBound$) on the number of nodes that a clusterhead handle. Therefore, none of the clusterheads are overloaded at any time. The clusterheads are chosen according their *weight* value: the higher weight a node gets, the better clusterhead it is. Amount of bandwidth, memory space or battery power of a processor could be used to determine weight values (the computation of weight values is out the scope of this paper).

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 to update or not their variables. 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 well-balanced clustering properties is presented in the section 3. Our algorithm is described in section 4. The self-stabilization proof is presented in section 5. Finally, the time complexity of our algorithm is analyzed in section 6.

2 Model

We model a distributed system by an undirected graph $G = (V, E)$ in which V is the set of nodes and E is the set of edges. There is an edge $(u, v) \in E$ if and only if u and v can directly communicate (u and v are said neighbors). The set of neighbors of a node $v \in V$ will be denoted by N_v. At node v in the network is assigned an unique identifier (ID). For simplicity, here we identify each node with its ID and we denote both with v. We assume the locally shared memory model of communication. Thus, each node v has a finite set of *local variables* such that the variables at a node v can be read by v and any neighbors of v, but can be only modified by v.

The code of each node v consists of a finite set of *rule*. A rule is a guarded statement of the form $Rule_i : Guard_i \rightarrow Action_i$, where $Guard_i$ is a boolean predicate involving the local variables of v and the local variables of its neighbors, and $Action_i$ is a program that modify

the local variables of v. $Action_i$ is executed by node v only if the $Guard_i$ is satisfied, in which case we say the node v is *enabled*.

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. In a *terminal* configuration, no node is enabled.

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 step of one or several enabled nodes. The nodes execute their programs - code asynchronously. Therefore, the subset of enabled nodes that do simultaneously their action during a computation step, is arbitrary chosen. A computation is *fair* if any node in G that is continuously enabled along a computation, will eventually perform an action. In this paper, we study only fair computation. A computation is *maximal* if it reaches terminal configuration or it is infinite. Let $\mathcal{C}$ be the set of possible configurations and $\mathcal{E}$ 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 $\mathcal{E}_B$.

In this paper, we use the notion *attractor* [11] to define self-stabilization.

Definition 1 ***(Attractor)**. Let B_3 and B_2 be subsets of $\mathcal{C}$. Then B_3 is an attractor from B_2 if and only if:*

***Convergence** - $\forall e \in \mathcal{E}_{B_2}, (e = c_1, c_2, ...),$*
$\exists i \geq 1 : c_i \in B_3.$

***Closure** - $\forall e \in \mathcal{E}_{B_3}, (e = c_1, c_2, ...), \forall i \geq 1, c_i \in B_3.$*

Observation 1 *Let B_1, B_2 and B_3 be subsets of $\mathcal{C}$.*
If B_3 is an attractor from B_2 and if B_2 is an attractor from B_1 then B_3 is an attractor from B_1.

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 from $\mathcal{C}$.*

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

3 Well-balanced Clustering for network

We consider weighted networks, i.e., a weight w_v is assigned to each node $v \in V$ of the network. In ad hoc sensor networks, amount of bandwidth, memory space or battery power of a processor could be used to determine weight values. The choice of the clusterheads will be 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. The nodes having an unique identifier (ID), one may replace the weight variable by the uplet $(weight, ID)$, to ensure that nodes have distinct weight values. Therefore, without losing generality, we assume that each node has a different weight.

Clustering means partitioning its nodes into *clusters*, each one with a *clusterhead* and (possibly) some *ordinary nodes*. In order to have well balanced clusters, the following *well-balanced clustering properties* have to be satisfied:

1. Every ordinary node always affiliates with one clusterhead of its neighborhood which has higher weight than its weight (*affiliation* condition).

2. There is at most $SizeBound$ nodes in a cluster (*size* condition).

3. If a clusterhead v has a neighboring clusterhead u such that $w_u > w_v$ then the size of u's cluster is $SizeBound$. (*clusterhead neighboring* condition).

The first requirement ensures that each ordinary node has direct access to its clusterhead (the head of the cluster to which it belongs), allowing fast intra and inter cluster communications. The first requirement also guarantees that each ordinary node affiliates with a suitable cluster (i.e., a cluster whose the head has a larger weight than its weight). The cluster management workload is proportional to the cluster size. Thus, the second requirement guarantees a clusterhead will be not overburden by the management workload of its cluster : each cluster has at most $SizeBound$ members. The third requirement limits the number of clusters. A node stays clusterhead only if cannot join an existing cluster : in its neighborhood, all suitable clusters are full (i.e., they have $SizeBound$ members).

4 Self-stabilizing Bounded Clustering algorithm

Algorithm 1 : Variables and Macro of on v

Constants

$w_v : \mathbb{N}$; - is the weight of node v

$SizeBound : \mathbb{N}$; - is the upper bound on a cluster size

Variables of node v

Ch_v: boolean; - indicate if v is or is not a clusterhead

$Head_v : IDs$; - is the clusterhead of v

$CD_v : \{IDs\}$ - is the list of nodes that can choose v as a their clusterhead. In the case v is an ordinary node, this list is empty.

$S_v : \mathbb{N}$ - is the size of v's cluster. If v is an ordinary node then $S_v = 0$.

Macros

v's neighbors could be clusterheads of v :

$$N_v^+ := \{z \in N_v : v \in CD_z \ \wedge \ Ch_z = T \ \wedge \ w_z > w_{Head_v} \ \wedge \ w_z > w_v\};$$

The size of v's cluster :

$$Size_v := |\{z \in N_v : Head_z = v\}|;$$

Computation of $CD2_v$:

begin
$CD0_v := \{z \in N_v : w_{Head_z} < w_v \ \wedge \ w_z < w_v\}$;
if $|CD0_v| \leq SizeBound - Size_v$ **then** $CD1_v := CD0_v$;
else $CD1_v$ contains the $SizeBound - Size_v$ smallest members of $CD0_v$;
if $CD_v \subseteq CD1_v \cup \{w \in N_v : Head_w = v\}$ **then** $CD2_v := CD1_v$;
else $CD2_v := \emptyset$;
end

The variables and macros are presented in Algorithm 1. Rules of the algorithm are pre-

sented in Algorithm 2.

The variable CD_v help us to guarantee the *size* condition (i.e., at most $SizeBound$ nodes are in the v's cluster): only the nodes of CD_v may integrate v's Cluster. V's Cluster is denoted $Cluster_v$. For each clusterhead v, we use the macro $CD2_v$ to set CD_v value. $CD0_v$ is the set of v's neighbor that want to enter in the v's cluster (i.e., their clusterhead is smaller than v). Only a subset of $CD0_v$ belongs to $CD2_v$ because we need to avoid that $|CD_v \cup Cluster_v| > SizeBound$. Otherwise, all nodes of CD_v faster than v would integrate the v's cluster before v action. After these moves, the v's cluster would no satisfy the *size* condition.

Notation 1
$Cluster_v := \{z \in N_v : Head_z = v\}$
$P_s(v) \equiv |CD_v \cup Cluster_v| \leq SizeBound.$

Algorithm 2 : Self-Stabilizing Clustering algorithm on v

Predicates

$\mathbf{G_0}(v) \equiv [(S_{Head_v} > SizeBound) \vee (Head_v \notin N_v)$
$\quad \vee (Ch_{Head_v} = F) \vee (w_{Head_v} < w_v)]$

$\mathbf{G_{11}}(v) \equiv (Ch_v = F) \wedge (N_v^+ = \emptyset) \wedge \mathbf{G_0}(v)$

$\mathbf{G_{12}}(v) \equiv (Ch_v = T) \wedge (N_v^+ = \emptyset) \wedge (Head_v \neq v)$

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

$\mathbf{G_2}(v) \equiv (N_v^+ \neq \emptyset)$

$\mathbf{G_3}(v) \equiv (Ch_v = T) \wedge [(S_v \neq Size_v) \vee (CD_v \neq CD2_v)]$

$\mathbf{G_4}(v) \equiv (Ch_v = F) \wedge [(S_v \neq 0) \vee (CD_v \neq \emptyset)]$

Rules

$\mathbf{R_1}(v) : \mathbf{G_1}(v) \rightarrow Ch_v := T;\ S_v := Size_v; CD_v := CD2_v;\ Head_v := v;$

$\mathbf{R_2}(v) : \mathbf{G_2}(v) \rightarrow Ch_v := F;\ S_v := 0;\ CD_v := \emptyset;$
$\quad Head_v := max_{w_z}\{z \in N_v^+\};$

$\mathbf{R_3}(v) : \neg\mathbf{G_1}(v) \wedge \neg\mathbf{G_2}(v) \wedge \mathbf{G_3}(v) \rightarrow S_v := Size_v;\ CD_v := CD2_v$

$\mathbf{R_4}(v) : \neg\mathbf{G_1}(v) \wedge \neg\mathbf{G_2}(v) \wedge \mathbf{G_4}(v) \rightarrow S_v := 0;\ CD_v := \emptyset$

Let us illustrate the computation of $CD2$ by an example. In the initial configuration of the Figure 1, the safety predicate is satisfied (i.e., every node v satisfied the predicate $P_s(v)$). In this configuration, the node having the biggest weight is node 15. All its neighbors want to enter its cluster $CD0_{15} = N_{15} - Cluster_{15} = \{3, 4, 7, 8\}$. Node 15 can allow only two nodes to enter in its cluster. Otherwise its cluster could have more than 3 members. Therefore $CD1_{15} = \{3, 4\}$. During the computation step where node 15 updates CD, all nodes of the current CD_{15} may and will integrate $Cluster_{15}$. If we set CD_{15} to $CD1_{15}$, we would have $|CD_{15} \cup Cluster_{15}| = |\{2, 3, 4, 7, 8\}| > 3$. To ensure that 15's cluster will surely satisfy the safety predicate after the computation step, CD_{15} is set to $\emptyset$.

The macro N_v^+ help us to guarantee the *clusterhead neighboring* condition. N_v^+ contains the list of v' neighbors that are better clusterheads for v than its current one. Node v will choose the node of N_v^+ having the highest weight as its clusterhead (rule R_2). The N_v^+ set contains

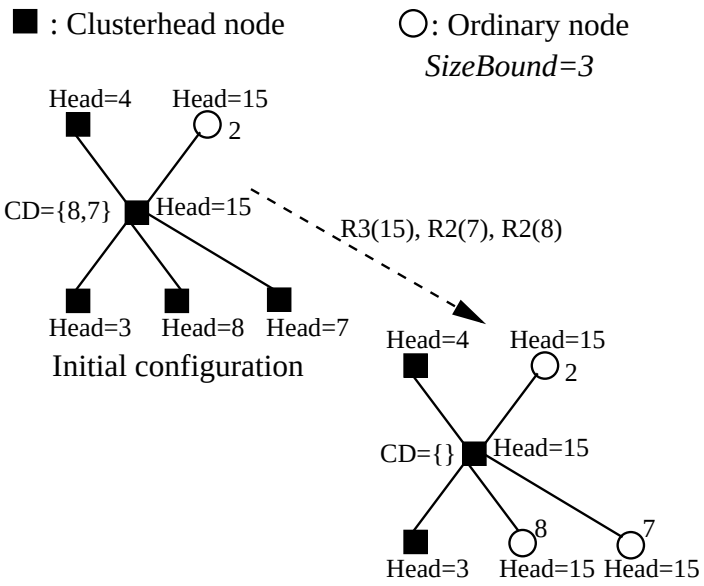

Figure 1: : Example of CD computation

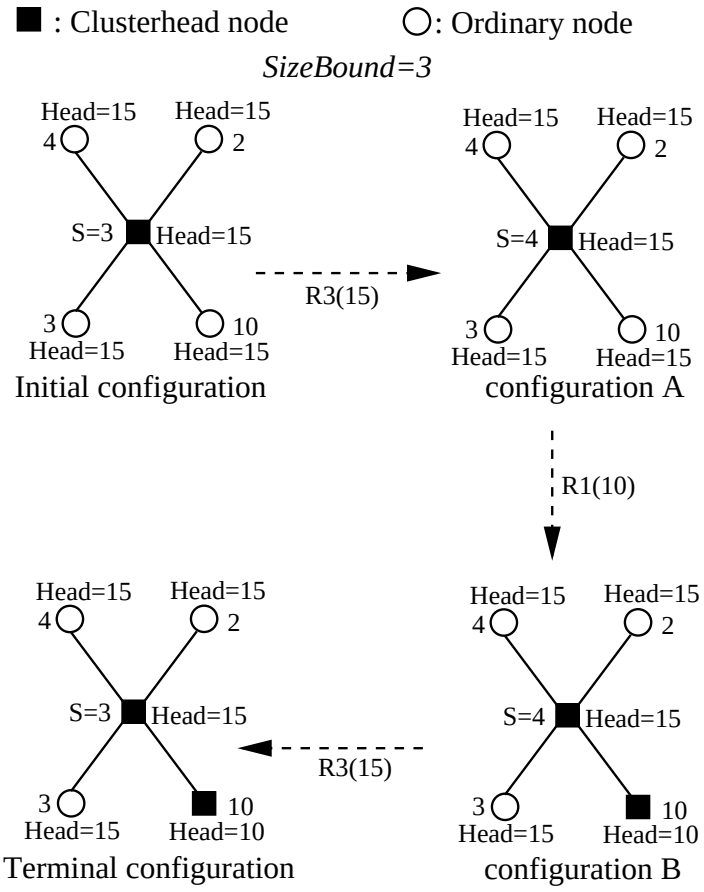

Figure 2: : Example of convergence

neighbors of v verifying the following properties (1) they are clusterhead, (2) their weight is bigger than the weight of the current clusterhead of v, (3) their weight is bigger than the weight of v, and (4) they accept that v enters in their cluster (i.e., v belongs to their CD set).

We split the possibles cases where a node v has to change its local variables according to the following mutually exclusive cases:

Case 1. v has to become a clusterhead (rule R_1). In v's neighborhood, there is not suitable clusterhead (i.e., N_v^+ is empty) and it belongs to a cluster having more than $SizeBound$ nodes **or** v does not satisfy the *Affiliation* condition. In this case, $G_{11}(v)$ is satisfied.

Case 2. v has to changed of cluster (rule R_2). The set of N_v^+ is not empty: v has in its neighborhood a more suitable clusterhead than its current one. In this case, $G_2(v)$ is satisfied.

Case 3. v has to change some local variable values without changing of cluster (it will stay clusterhead or ordinary). If v is a clusterhead then $G_3(v)$ or $G_{12}(v)$ is satisfied. If v is an ordinary node then $G_4(v)$ is satisfied.

Algorithm 2 is illustrated in Figure 2, in this example $SizeBound = 3$. Initially, there is one cluster having 4 members (the *size* condition is not verified). The node 10 will quit this cluster, and will form a new cluster (action R_1), once the only clusterhead (node 15) has updated its S variable (action R_3).

5 Proof of self-stabilization

In subsection 5.1, we prove that A_2 is an attractor from $\mathcal{C}$. Thus, from any safe configuration, the safety predicate holds continuously until the protocol converges to a legitimate configuration. Such an example is given in figure 3. Initially all clusters have less than 3 members, this property is verified along any computation reaching a legitimate configuration.

Definition 3 *Let A_2 be the set of safe configurations defined by $\{\mathcal{C} \mid \forall v : P_s(v)$ is satisfied $\}$.*

A legitimate configuration is a terminal configuration where every node v satisfies $P_s(v)$ and $CD_v = \emptyset$.

In subsection 5.2, we prove that along any fair computation, a legitimate configuration is reached from any configuration.

In subsection 5.3, we establish that a legitimate configuration verifies the well-balanced clustering properties.

5.1 Safety

Notation 2 *Let c be a configuration. We denote $CD_v(c)$ the value of the CD variable of the node v in c. We denote $Cluster_v(c)$ the value of the $Cluster$ variable of the node v in c.*

Observation 2 *Assume that we have a computation step $c_1 \stackrel{cs}{\rightarrow} c_2$. According to the macro N^+ and to the rule R_2, $Cluster_v(c_2) \subseteq (Cluster_v(c_1) \cup CD_v(c_1))$.*
Assume that v updates CD_v during the computation step cs. According to the macro $CD2_v$,

- $CD_v(c_2) = \emptyset$ *or*
  $|CD_v(c_2) \cup Cluster_v(c_1)| \leq SizeBound$ *(i.e. $CD1_v(c_1) = CD_v(c_2)$)*
- $CD1_v(c_1) \cap Cluster_v(c_1) = \emptyset$.

Notation 3
$P_s(v) \equiv |CD_v \cup Cluster_v| \leq SizeBound.$
$P_t(v) \equiv CD_v = \emptyset.$

Lemma 1 *A_2 is closed.*

Proof: Assume that: (1) we have a configuration c_1 in which $P_s(v)$ holds and (2) we have a computation step $c_1 \stackrel{cs}{\rightarrow} c_2$ reaching a configuration not satisfying $P_s(v)$. We will prove that is a contradiction.
There exists at least a node z such that :
$z \notin CD_v(c_1) \cup Cluster_v(c_1)$ and
$z \in CD_v(c_2) \cup Cluster_v(c_2)$.
During cs, v have modified CD_v to include z: $z \in CD_v(c_2)$ (see Observation 2); thus $CD_v(c_2) \neq \emptyset$. We have $|CD_v(c_2) \cup Cluster_v(c_1)| \leq SizeBound$.
Case 1: $Cluster_v(c_2) \subseteq Cluster_v(c_1)$. $P_s(v)$ is satisfied in c_2. That is a contradiction.
Case 2: $Cluster_v(c_2) = \{u_1, .. u_m\} \cup Cluster_v(c_1)$. According to the observation 2, we have $\{u_1, .. u_m\} \subset CD_v(c_1)$.
According to the $CD2_v$ definition, $CD_v(c_1) \subset (CD_v(c_2) \cup Cluster_v(c_1))$.
We have $|CD_v(c_2) \cup Cluster_v(c_1)| \leq SizeBound$.
Thus $|CD_v(c_2) \cup \{u_1, .. u_m\} \cup Cluster_v(c_1)| \leq SizeBound$.
We conclude that $P_s(v)$ is satisfied in c_2. That is a contradiction. □

First, we establish that A_1 is an attractor from $\mathcal{C}$, then we prove that A_2 is an attractor from A_1, in order to conclude that A_2 is an attractor from $\mathcal{C}$.

Lemma 2 *$A_1 = \{\mathcal{C} \mid \forall v : P_s(v) \vee P_t(v)$ is satisfied $\}$ is closed.*

Proof:
Assume that we have a computation step cs, $c_1 \stackrel{cs}{\rightarrow} c_2$ such that in c_1, $P_s(v) \vee P_t(v)$ is satisfied; and $P_s(v) \vee P_t(v)$ is satisfied in c_2. As $P_s(v)$ is closed, in c_1, $P_s(v)$ is not verified. We conclude that $P_t(v)$ is verified in c_1 and $P_s(v) \vee P_t(v)$ is not satisfied in c_2 : v updates the value CD_v during cs.
Case 1: v is a clusterhead.
According to the observation 2, we have $|CD_v(c_2) \cup Cluster_v(c_1)| \leq SizeBound$ or $CD_v(c_2) = \emptyset$. In the first case, $P_s(v)$ is satisfied in c_2 (there is a contradiction). In the latest case, we have $CD_v(c_2) = \emptyset$, thus, $P_t(v)$ is still verified in c_2 (There is a contradiction).
Case 2: v is a ordinary node. After any update of CD_v, we have $CD_v = \emptyset$, thus $P_t(v)$ is satisfied. There is a contradiction. □

Lemma 3 *Let a configuration c in which $P_s(v) \vee P_t(v)$ is not satisfied. v is enabled at c.*

Proof:
Assume that v is an ordinary node. Till $P_t(v)$ is not verified $R_4(v)$ is enabled. Assume that v is a clusterhead. In c, we have $|CD1_v \cup Cluster_v| = max(|Cluster_v|, SizeBound)$ and $CD_v \neq \emptyset$.
If $CD_v \cap (CD1_v \cup Cluster_v) \neq CD_v$ then $CD2_v = \emptyset$. Thus, $G_3(v)$ is satisfied.
If $CD_v \cap (CD1_v \cup Cluster_v) = CD_v$ then
$CD2_v = CD1_v$. As $P_s(v)$ is not satisfied, $|CD_v \cup Cluster_v| > SizeBound$. Thus, $|CD1_v \cup Cluster_v| > SizeBound$. We conclude that $|CD1_v \cup Cluster_v| = |Cluster_v|$, thus $|CD1_v| = 0$ (because $CD1_v \cap Cluster_v = \emptyset$). Therefore, $G_3(v)$ is satisfied because $CD_v \neq (CD2_v = \emptyset)$.
If $G_3(v)$ is satisfied then $R_1(v)$, $R_2(v)$ or $R_3(v)$ is enabled. □

Lemma 4 *Let A_1 be the set of configurations defined by $\{\mathcal{C} \mid \forall v : P_s(v) \vee P_t(v)$ is satisfied $\}$. Any computation of $\mathcal{E}$ reached a configuration of A_1.*

Proof: Assume that a computation *seq* not reaching a configuration of A_1 exists. As the predicate $P_s(v) \vee P_t(v)$ is closed. There is a node v that never satisfying the predicate $P_s(v) \vee P_t(v)$ along the computation *seq*. Along the computation *seq*, v is always enabled (lemma 3), thus v will infinitely often perform a rule. Let *cs* be a computation step of the *seq* where v perform an action : $c_1 \stackrel{cs}{\rightarrow} c_2$. According to the assumption, the predicate $P_s(v) \vee P_t(v)$ is not satisfied in c_2.
Any rule performed by v updates CD_v. According to the observation 2, we have $CD_v(c_2) = \emptyset$ or
$|CD_v(c_2) \cup Cluster_v(c_1)| \leq SizeBound$.
In the first case, c_2 satisfies $P_t(v)$. The second case is impossible, because
$CD_v(c_1) \subset (CD1_v(c_1) \cup Cluster_v(c_1))$
$CD1_v(c_1) = CD_v(c_2)$
$|CD_v(c_1) \cup Cluster_v(c_1)| > SizeBound$.

$P_s(v) \vee P_t(v)$ is satisfied in c_2; there is a contradiction. □

Lemma 5 *Any computation of $\mathcal{E}_{A_1}$ reached a configuration of A_2.*

Proof: In a configuration of A_1, but not of A_2, there is node v such that $P_t(v) \wedge \neg P_s(v)$ is satisfied. Till $P_s(v)$ is not satisfied, $P_t(v)$ is satisfied because A_1 is closed. Thus, no node can integrate $Cluster_v$.
Case 1: Assume that v is an ordinary node. $Cluster_v \neq \emptyset$. We focus on a node u of $Cluster_v$. Because $Ch_v = F$, $G_{11}(u)$ or $G_2(u)$ is satisfied forever. By fairness, u eventually performs the rule $R_1(u)$ or the rule $R_2(u)$. After the u's action, $u \notin Cluster_v$.
Thus, $|Cluster_v| = 0 \leq SizeBound$ eventually holds.
Case 2: Assume that v is a clusterhead.
We have $Size_v > SizeBound$. If $S_v \leq SizeBound$ then $G_3(v)$ is satisfied. v eventually performs the rule R_3. After the v's action, $S_v = Size_v > SizeBound$ holds till $|Cluster_v| = Size_v > SizeBound$. We consider a node u of $Cluster_v$. $G_{11}(u)$ or $G_2(u)$ is satisfied till $Size_v > SizeBound$, because $S_{Head_u} > SizeBound$. By fairness, u eventually performs the rule $R_1(u)$ or the rule $R_2(u)$. After this action, $Size_v$ decreases, because $u \notin Cluster_v$. Eventually, we reach a configuration where $|Cluster_v| = Size_v \leq SizeBound$. □

Corollary 1
$A_s = A_2 \cap \{c \in \mathcal{C} \mid \forall v : |Cluster_v| \leq SizeBound\}$ *is an attractor from $\mathcal{C}$.*

Following directly from Lemma 1 and 5, A_2 is an attractor from A_1. Following directly from Lemma 2 and 4, A_1 is an attractor from $\mathcal{C}$. Thus A_2 is an attractor from $\mathcal{C}$. $A_2 = A_s$, because in A_2, any node v verified the following predicate $|Cluster_v| \leq |Cluster_v \cup CD_v| \leq SizeBound$.
□

5.2 Convergence

Lemma 6 $P_g \equiv (G_{12}(v) = F) \wedge (G_4(v) = F)$.
$A_3 = A_s \cup \{c \in \mathcal{C} \mid \forall v : P_g(v)$ *is satisfied*$\}$ *is an attractor from $\mathcal{C}$.*

Proof: If v verifies the predicate $G_{12}(v)$ then v is enabled and will stay enabled up to the time where v performs $R_1(v)$. As all computations are fair, v eventually performs $R_1(v)$.

After this action $G_{12}(v)$ is never satisfied.
If v verifies the predicate $G_4(v)$ then v is enabled and will stay enabled up to the time where v performs $R_4(v)$. As all computations are fair, v eventually performs $R_4(v)$. After this action $G_4(v)$ is never satisfied. □

Lemma 7 *$A_4 = A_3 \cup \{c \in \mathcal{C} \mid \forall v : Head_v \in N_v \cup \{v\}\}$ is an attractor from $\mathcal{C}$.*

Proof: In A_3, if $N_v^+ = \emptyset$ then $G_1(v)$ is satisfied. As all computations are fair, v eventually performs $R_1(v)$: we have $Head_v = v$.
In A_3, if $N_v^+ \neq \emptyset$ then $G_2(v)$ is satisfied. As all computations are fair, v eventually performs $R_2(v)$; we get $Head_v \in N_v$.
Once $Head_v \in N_v \cup \{v\}$ is satisfied, it is never falsified (see the rule actions). □

The convergence is done in step. At the end of the i^{th} step the configuration set $L"_i$ is reached : all nodes of Set_i has chosen forever their clusterhead. We define Set_i and $L"_i$ (for any value of i) as follows.

Notation 4

$L"_0 = A_4$ *and* $Set_0 = \emptyset$.

V_i *is the set of nodes that do not belong to* Set_i :

$$V_i = V - Set_i.$$

vh_i *is the node with the highest weight in* V_i.

$L_{i+1} = L"_i \cap \{c \in \mathcal{C} \mid Ch_{vh_i} = T\}$

$SizeBound_i$ *is the value* $min(SizeBound, |N_{vh_i} \cap V_i|)$.

$L'_{i+1} = L_{i+1} \cap \{c \in \mathcal{C} \mid |Cluster_{vh_i}| = SizeBound_i\}$.

$Set_{i+1} = Set_i \cup \{vh_i\} \cup Cluster_{vh_i}$.

$L"_{i+1} = L'_{i+1} \cap \{c \in \mathcal{C} \mid \forall v \in Set_{i+1} : CD_v = \emptyset\}$.

The convergence steps are illustrated in figure 3 where the weight of nodes are ordered as the following: $X_{i-1} > Y_i > X_i > Y_{i+1}$. Initially A_4 is reached. After the first computation step, L_1 is reached (the node having the largest weight, X_0, is a clusterhead). After the second computation step L'_1 is reached ($Cluster_{X_0} = \{Y_1\} = N_{X_0}$). After the third computation step, L_2 is reached, X_1, the node having the largest weight of $V - \{X_0, Y_1\}$, is a clusterhead. After the fourth computation step, L'_2 is reached ($Cluster_{X_1} = \{Y_2\} = N_{X_1} - \{X_0, Y_1\}$).

Observation 3 *Let v be a node of V_i.*
We have, by definition of V_i, $Head_v \notin Set_i$.
If $Set_i \neq V$ then ($Set_i \subset Set_{i+1}$) and ($Set_i \neq Set_{i+1}$).

At each step, Set_i increases up to contain all nodes. Once $Set_i = V$, we will prove that a legitimate configuration is reached.

Lemma 8 *For any value of i, L_{i+1} is an attractor from $\mathcal{C}$, assuming that $L"_i$ is an attractor from $\mathcal{C}$.*

Proof: vh_i is the node with the biggest weight in V_i. Let z be in N_{vh_i}. If $z \in Set_i$, we have $CD_z = \emptyset$ (see definition of $L"_0$ or $L"_{i+1}$). So $N_{vh_i}^+$ is empty: vh_i never executes $R_2(vh_i)$.
According to the observation 3, we have $Head_{vh_i} \in V_i$, thus by definition of vh_i, $w_{Head_{vh_i}} \leq w_{vh_i}$.
If vh_i is not a clusterhead then $G_1(vh_i)$ is satisfied because $w_{Head_{vh_i}} < w_{vh_i}$. As all computations are fair, vh_i eventually performs $R_1(vh_i)$. After that $Ch_{vh_i} = T$ and $Head_{vh_i} = vh_i$, forever. □

Lemma 9 *Let c_1 be a configuration of L_{i+1}. Let cs be a computation step from c_1: $c_1 \stackrel{cs}{\rightarrow} c_2$. $Cluster_{vh_i}(c_1) \subseteq Cluster_{vh_i}(c_2)$.*

Proof:
Let u be a node of $Cluster_{vh_i}$ (i.e., $Head_u = vh_i$). In L_{i+1}, any neighbor z of u such that $w_z > w_{vh_i}$ is in Set_i: $CD_z = \emptyset$. Thus $z \notin N_u^+$; we conclude that N_u^+ is empty forever. Thus, $G_2(u)$ is never enabled. In L_{i+1}, $G_1(u)$ is also not enabled because $[(S_{Head_u} \leq SizeBound) \wedge (Head_u \in N_u) \wedge (Ch_{Head_u} = T) \wedge (w_{Head_u} > w_u)]$. We conclude that the node u stays in the Cluster of vh_i forever. □

Lemma 10 *Let c_1 be a configuration of L_{i+1}. Let cs be a computation step from c_1: $c_1 \stackrel{cs}{\rightarrow} c_2$. We have $CD1_{vh_i}(c_2) \neq CD1_{vh_i}(c_1)$ if only if $Cluster_{vh_i}(c_1) \neq Cluster_{vh_i}(c_2)$.*

Proof:
In any configuration c_1 of L_{i+1}, we always have $CD0_{vh_i}(c_1) = N_{vh_i} \cap V_i - Cluster_{vh_i}(c_1)$; and $CD1_{vh_i}(c_1)$ contains the $SizeBound - |Cluster_{vh_i}(c_1)|$ smallest members of $CD0_{vh_i}(c_1)$. We have $CD1_{vh_i}(c_2) \neq CD1_{vh_i}(c_1)$ only if one of the following cases happens:

- a node of $Cluster_{vh_i}(c_1)$ changes of clusterhead during cs. It is impossible see lemma 9.
- a node u integrates $Cluster_{vh_i}$ during cs. We have $Cluster_{vh_i}(c_1) \neq Cluster_{vh_i}(c_2)$.

□

Lemma 11 *For any value of i, L'_{i+1} is an attractor from $\mathcal{C}$ assuming that L_{i+1} is an attractor from $\mathcal{C}$.*

Proof: Once L_{i+1} is reached, we have $Ch_{vh_i} = T$ and $Head_{vh_i} = vh_i$. Only the nodes of V_i may be in the Cluster of vh_i, therefore, we have $|Cluster_{vh_i}| \leq SizeBound_i$.
Assume that size of $Cluster_{vh_i}$ is forever smaller than $SizeBound_i$. As no node can quit $Cluster_{vh_i}$ (lemma 9), $Cluster_{vh_i}$ will eventually stay identical forever. According lemma 10, $CD1_{vh_i}$ will eventually stay identical forever.
$CD1_{vh_i}$ contains $SizeBound_i - |Cluster_{vh_i}|$ nodes.

Till $CD_{vh_i} \neq CD1_{vh_i}$, $R_3(vh_i)$ is enabled. By fairness, $R_3(vh_i)$ action will be eventually performed. After $R_3(vh_i)$ action, we have $CD_{vh_i} = \emptyset$ or $CD_{vh_i} = CD1_{vh_i}$ (now, $CD_{vh_i} \subset (CD1_{vh_i} \cup Cluster_{vh_i})$). If $CD_{vh_i} \neq CD1_{vh_i}$ $R_3(vh_i)$ is still enabled forever. By fairness, $R_3(vh_i)$ action will be again performed. After $R_3(vh_i)$ action, we have $CD_{vh_i} = CD1_{vh_i}$. We conclude that any computation reaches a configuration where $CD_{vh_i} = CD1_{vh_i} \neq \emptyset$. Now, $R_3(vh_i)$ is never enabled (i.e., CD_{vh_i} is always equal to CD_{vh_i}).

Le u be a node of CD_{vh_i}. By definition of vh_i and V_i, we have $w_{Head_u} < w_{vh_i}$ and $w_u < w_{vh_i}$, thus $vh_i \in N_u^+$. Any neighbor z of u such that $w_z > w_{vh_i}$ is in Set_i by definition of vh_i. Thus $CD_z = \emptyset$. Therefore vh_i is the node of N_u^+ having the biggest weight. $R_2(u)$ is enabled forever. By fairness, $R_2(u)$ action will be eventually performed: u will choose vh_i as its clusterhead. There is a contradiction, $Cluster_{vh_i}$ does not stay identical forever. □

Lemma 12 *For any value of i, L"$_{i+1}$ is an attractor from $\mathcal{C}$, assuming that L'_{i+1} is an attractor from $\mathcal{C}$.*

Proof: Let v be a node of Set_{i+1} that does not belong to Set_i. If v is an ordinary node, $CD_v = \emptyset$ because A_3 is a subset of L"$_{i+1}$. If v is a clusterhead then $v = vh_i$. By definition of L'_{i+1}, $\forall u \in N_v : w_{Head_u} \geq w_v$ or $|Cluster_v| = SizeBound$. Thus $CD2_v = \emptyset$, in L'_{i+1}. If $CD_v \neq \emptyset$, then $R_3(v)$ is enabled forever. Once the rule is executed, $CD_v = \emptyset$ holds. □

Theorem 1 *Let A_5 a configurations set defined by $A_5 = A_4 \cap \{c \in \mathcal{C} \mid \forall v : CD_v = \emptyset\}$. The system eventually reaches a terminal configuration of A_5.*

Proof: According to the Observation 3, $Set_i \subset Set_{i+1}$. Thus, there exists j such that $Set_j = V$.
$L"_j$ is an attractor because $L"_0$, L_i, L'_i, and $L"_i$ are the attractors for any value of $i \leq j$. In $L"_j$, the rule R_1, the rule R_2, and the rule R_4 are not enabled on any node. Only the rule R_3 may be enabled forever, on a node v. By fairness, v will execute $R_3(v)$, then v is never enabled. We conclude that a terminal configuration of L''_i will be reached. Any configuration of L''_i belong to A_5. □

5.3 Correctness

Theorem 2 *Once a terminal configuration of A_5 is reached, the well-balanced clustering properties are satisfied.*

Proof: In a terminal configuration of A_5, for every node z, we have $G_i(z) = F : i = 1..4$ and $CD_z = \emptyset$ (see theorem 1).
Case 1. z is an ordinary node.
$G_{11}(z) = F$ implies $(Head_z \in N_v) \wedge (Ch_{Head_z} = T)$ and $(w_{Head_z} > w_z)$. Thus, z satisfies *affiliation* condition.
Case 2. z is a clusterhead node. Following Corollary 1, in a terminal configuration $S_z \leq SizeBound$, thus, the *size* condition is satisfied.
Assume that there exists a node v such that $v \in N_z : (Ch_v = T) \wedge (w_v > w_z) \wedge (Size_v < SizeBound)$. In this case, according to the macro $CD0_v$, we have $z \in CD0_v$, because $w_{Head_z} = w_z < w_v$. As $Size_v < SizeBound$, according to the macro $CD1_v$, we have $CD1_v \neq \emptyset$. We also have $CD_v = \emptyset$ then $G_3(v) = T$ (this is a contrary). We conclude that every clusterhead v in z neighborhood verifies $(w_v \leq w_z)$ or $(S_v = SizeBound)$. Thus the *clusterhead neighboring* condition is satisfied. □

6 Convergence times

In this section, we compute the time needed to reach a safe configuration and the time needed to reach a legitimate configuration. We consider synchronous computation, in which every process performs its code simultaneously. Thus, all enabled process perform a rule during each computation step.

Studying proofs that A_s is an attractor, help us to establish the number of synchronous computation steps needed to reach a safe configuration. A node v, that does not satisfy $P_s(v) \vee P_t(v)$, is enabled (see lemma 3) after any action of v, $P_s(v) \vee P_t(v)$ is satisfied (see the proof of lemma 4). Thus, A_1 is reached after one synchronous computation step. Two more synchronous computation step are needed to reach A_2. After a synchronous computation step, if $Size_v > SizeBound$ then $S_v > SizeBound$. Thus, all ordinary nodes u such that $Size_{Head_u} > SizeBound$, are enabled. During the third synchronous computation steps these nodes u will quit their cluster (proof of lemma 5). Therefore, a safe configuration is reached, after at most 3 synchronous computation steps (proof of corollary 1).

The stabilization time is the maximum number of computation steps needed to reach a stabilized configuration from an arbitrary initial one. Figure 3 presents a scenario to measure stabilization time, in the worst case: the initial configuration is the worst one. In this configuration (Figure 3.a), there are $(N-1)/2$ clusters where N is the network size. Each cluster C_i $(i > 1)$ includes a clusterhead Y_i and a ordinary node X_{i+1}. The weight of nodes are

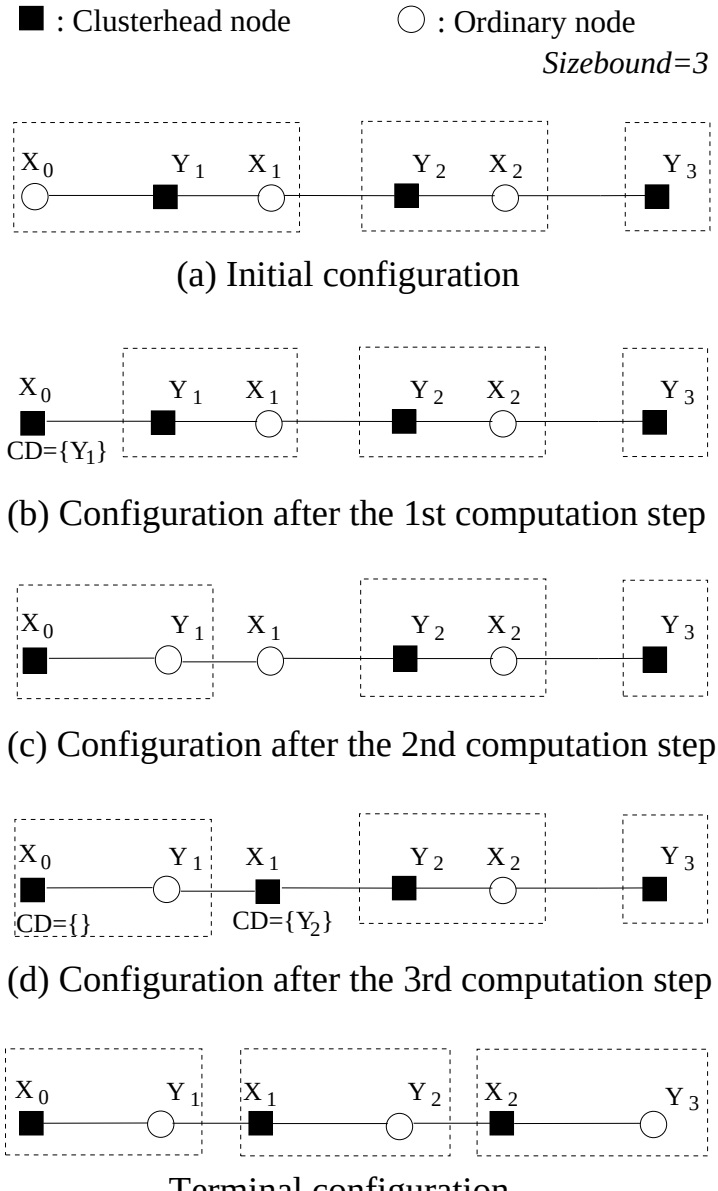

Figure 3: Stabilization time.

ordered as the following: $X_{i-1} > Y_i > X_i > Y_{i+1}$. Initially, only the node X_0 is enabled. X_0 is not a clusterhead, and its has the largest weight of the system; thus $G_{11}(X_0)$ is satisfied. In the first round X_0 performs the rule R_1. Now, Y_1 is enabled, because $N^+_{Y_1} = \{X_0\}$; the other nodes are still not enabled. Thus, during the second round, the node Y_1 performs the rule R_2 to integrate X_0's cluster. Then, X_1 is enabled; the $G_{11}(X_1)$ predicate is satisfied because $Cl_{Head_{X1}} = F$. In the third round, X_1 performs the rule R_1, and so one. All nodes will update their status; during a computation step, only one node changes its status.

The stabilization time is $O(N)$.

References

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

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

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

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

[5] 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.

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

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

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

[9] C. Johnen and L. H. Nguyen. Robust self-stabilizing clustering algorithm. In *Proceedings of the 10th International Conference On Principles Of Distributed Systems, Springer LNCS 4305 (OPODIS'06)*, pages 408–422, 2006.

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

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

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

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

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

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